A Topology Space Odyssey: Neural Topological Evolution

Tingwu Wang
University Of Toronto & Vector Institute
Contents

1. Introduction
   a. The Problem of Robotics Structure Search
   b. Neural Architecture Search and Evolutionary Strategy

2. Algorithm
   a. Topology Representation
   b. NerveNet and NerveNet++: Policy Representation with Graph
   c. Rapid Evolving using Policy Inheritance with Amortized Fitness
   d. Neural Topological Pruning
   e. Summary

3. Experiments
   a. Evolution Topology Search
   b. Fine-tuning Species
   c. Neural Topology Pruning and Misc

4. References
The Problem of Robotics Structure Search

1. Assume you are an engineer in a robotic company, given the same computation budget & time, what's the best product you could deliver to the public?

2. The search space of the structures (or topologies) is infinite
   Using human intuition on the structures (or topologies) of robot
   1. Robustness
   2. Capability
   3. Cost
   4. Easy to control
   5. ...

[Image of robots]
The Problem of Robotics Structure Search

3. How do we evaluate the topology?
   1. Model based control?
      1. Slow (7x real time on iLQR with MPC [1]).
      2. Heavy engineering of dynamics
      3. Performance is heavily dependent on the quality of dynamics
   2. Model free control?
      1. Fast to test, but >> 1 million timesteps needed before convergence [2, 3].
      2. Policy not transferable between topologies [4].
   3. In practice: complex engineering and evaluation of controllers are needed.
      1. Bigdog was invented 2005 [5] and has been polished until now.

4. Key to the problem
   1. Outer-loop: **searching** over the topology space.
   2. Inner-loop: **evaluating** the topology.
Neural Architecture Search and Evolutionary Strategy

1. Our problem is more related to Neural Architecture Search (NAS) [6] than Evolutionary Strategy.
   1. Outer-loop: different potential architectures (e.g. ResNet, VGG)
   2. Inner-loop: train the network.

2. Evolutionary Strategy (ES) [7] or Genetic Algorithms [8]
   2. Inner-loop: random search of policy weights.
1. Introduction
   a. The Problem of Robotics Structure Search
   b. Neural Architecture Search and Evolutionary Strategy
2. Algorithm
   a. Topology Representation
   b. NerveNet and NerveNet++: Policy Representation with Graph
   c. Rapid Evolving using Policy Inheritance with Amortized Fitness
   d. Neural Topological Pruning
   e. Summary
3. Experiments
   a. Evolution Topology Search
   b. Fine-tuning Species
   c. Neural Topology Pruning and Misc
4. References
Before we do guided search, to represent the topologies, every species is associated with the topology graph $G=(V, E)$ and node attributes $\{a(u)|u \in V\}$ (related to how we train the network).

1. Node: bodies of the robot (fins, fish-torso, ...)
2. Edge: the joints between the nodes (the joint between the fish-torso and fin)

3. Attribute:
   1. Size of body
   2. Relative position
   3. Joint information
Topology Representation

1. Primitive operations on the graphs during topology search
   1. Add-Node
   2. Add-Graph
      1. To allow faster evolution and reuse the sub-trees in the graph with good functionality.
      2. Randomly mirror the attribute to incorporate a symmetry prior.
   3. Del-Graph
   4. Pert-Node
   1. Input model
      For each node (0, 1, 2, 3, 4), we fetch the corresponding observation from the input features.
   2. Propagation model
      Let each node send messages to its neighbours, aggregate messages and update the node’s hidden states
   3. Output model
NerveNet and NerveNet++: Policy Representation with Graph

1. NerveNet improves the transferability
   1. We are able to transfer the learnt policy even the topologies are different. The weight vector $\theta$ has exactly the same shape

2. Further proposed NerveNet++, which passes states to the next time step and utilizes truncated back-propagation.
Rapid Evolving using Policy Inheritance with Amortized Fitness

1. A standard loop of topology search
   1. Init a list of topologies (each species has one topology, making a generation)
   2. Evaluate the fitness of topologies and eliminate the ones with worst fitness
   3. Generate new species (a new generation)

2. We have to balance between evolving new species (with new topologies) and evaluating the species (train network).
   1. More time spent on evolving?
      Less computation resource on evaluating, and the estimated fitness is less accurate.
   2. More time spent on evaluating?
      Less computation resource on evolving, and the search space is limited.
Rapid Evolving using Policy Inheritance with Amortized Fitness

1. With NerveNet, we are moving towards unifying the evolving and evaluating.
   1. Instead of using fitness, we use amortized fitness
   2. Amortized fitness is the intermediate fitness
   3. Spreads the computation cost across generations (continue the evaluating in next generation)
2. Using Policy Inheritance from parent to children species.

2. We estimate the fitness using amortized fitness.
   1. It is biased, but is refined with generation
   2. New species will not be dominated by existing old species (NerveNet’s transferability).
Neural Topological Pruning

1. One unanswered question: how do we generate the new species?
   1. Random mutations from current generation, with a predefined proposal distribution?
   2. Train a proposal network that decide what topology (species) to generate?

2. Random proposal with Neural Topological Pruning (NTP)
   1. We generate lots of candidates around the current generation
   2. Using a pruning network to remove unpromising ones
Neural Topological Pruning

1. What is a good pruning network?
   1. Avoiding wasting computation resource on species with low expected fitness.
   2. Encouraging the model to test species of which it is not certain about.

2. We use a GNN that’s similar to NerveNet policy network, and replace input with only node attributes.
   1. Bayesian optimization framework to balance the trade-off between exploration vs exploitation (model uncertainty).
   2. We follow sample from the model posterior

\[
G^* = \arg \max_G \mathbb{E}_{P(\theta_G|D_\xi)} \left[ V_\xi (G, \{a(u)\}| \theta_G) \right].
\]

1. Using "Dropout as a bayesian approximation" [10]
Summary

Algorithm 1 Neural Topological Evolution

1: Initialize $\mathcal{N}$ species with weights and topology $\{\theta_i, G_i\}$  
2: \textbf{while} True \textbf{do} \hfill $\triangleright$ Evolution outer loop
3: \hspace{1em} \textbf{for} species $i$ alive \textbf{do} \hfill $\triangleright$ Species fitness inner loop
4: \hspace{2em} Train and evaluate Amortized Fitness $\xi_i$ of the species using NerveNet++.  
5: \hspace{1em} \textbf{end for}
6: \hspace{1em} Eliminate $\beta\mathcal{N}$ species with the worst fitness score \hfill $\triangleright$ Selection scheme
7: \hspace{1em} Mutate new species with Policy Inheritance \hfill $\triangleright$ Mutation
8: \hspace{1em} Neural Topology Pruning \hfill $\triangleright$ Prune off the non-promising species
9: \textbf{end while}
1. Two Environments: Fish & 2d-walker

(a) We observed that the fish agents have learnt to grow symmetric side-fins over generations.

(b) Our walker species gradually grows two foot-like structures from randomly initialized body graph.
Evolution Topology Search

1. Two Baselines.

(a) Results on fish environment.

(b) Results on walker environment.

Figure 2: The performance of the topology search for Brute-force, MLP and NTE.
Fine-tuning Species

1. We test the fine-tuning on three species given the default structure. All the methods are given equal computation resource.

(a) Fine-tuning fish3d. (b) Fine-tuning leg-walker. (c) Fine-tuning cheetah.
Neural Topology Pruning Results and Misc

1. We show the result with and without the NTP and using different number of cores.
Reference