Notes for A. Sverdlov's Presentation of *Efficient Reinforcement Learning through Evolving Neural Network Topologies*
Given on May 11th, 2004
Todd W Flyr

This presentation was broken in to three main segments:
Neural Networks
Neural Evolution
Neural Evolution of Structure

Neural Networks

A neural network is a network of weighted nodes, typically composed of an input layer, one or more hidden layers, and an output layer.

Neural Networks have been shown to be universal function approximators.

As a function approximator, a neural network needs to be trained, but traditional approach is backpropagation, which is not a good way of finding a global maximum.

Evolutionary algorithms are a method of unsupervised learning that involve a concept of genes (genotype—what the gene encodes for and phenotype—what is expressed in the organism (agent)), agents which are evaluated for fitness (similar to an RL reward), a degree of randomization for create a robust collection of agents to be evaluated, and populations of agents to be evaluated.ß

The problem space may not be the correct one populated by the networks.

Neural Evolution
This paper addresses the issues of evolving not only the free parameters of a fixed neural network, but also evolving the structure of such a network.
To do neural evolution:

- randomly cross breed the genomes of the networks
- introduce mutations
- apply fitness function to eliminate weak breeds

repeat (until a candidate network meets the fitness criteria)

Neural Evolution of Structure
Begin by picking a network topology—
If you start with a large network, large populations are large networks are "hard to converge" to a reasonable solution in a feasible amount of time.
It is known that the number of hidden nodes plays a huge role in whether the network converges (too few and it cannot approximate the solution space accurately enough to converge and too many and it takes too long to converge).

Thus to solution is to have an algorithm that can augment the topology—i.e. modify neural network structure as well as weights—
Some issues with such an algorithm:
-can a genetic representation of network parameters and structure allow for topologies to crossover in a meaningful way?

-"how can a network that needs only a few generations to optimize be protected so that it does not disappear from the population prematurely?"

-"how can topologies be minimized throughout evolution without the need for a specially contrived fitness function that measures complexity?"

The solution:
Create *Node Genes* and *connection genes*
A Node Gene—consists of a sensor, and output, and a weight.
A Connection Gene refers to two node genes being connected—the weight—from out of one node and into another—and the *innovation #*.

A Structural mutation—may add a connection where none existed before or may add a node (which is done by finding a connection to split, replacing the split connection by inserting a node and two new connections

Crossover is achieved via matching up genes by innovation #--
The offspring is a gene formed by merging nodes with the same innovation # from a more fit parent. The innovation number marks the birth of a gene and allows for proper matching of genes across networks of a diverse population.

The excess and disjoint genes are added from the more fit parent—see figure 3 in paper, otherwise, if both parents are equally fit, then both contribute genes to the offspring.

Protecting Innovation Via Speciation
This preserves diversity—and allows certain networks that solve certain subsets of problems to preserve their innovations without being lost due to perhaps a poorer overall fitness.

Pole Balancing
The authors of the paper used pole balancing to experiments to test their algorithms.
Single pole balancing was considered too easy, so they used double pole balancing.

They were able to evolve a network that performed double pole balancing. It wasn't better at the task, but it required less hidden nodes (4) (when velocity information was provided).

For double pole balancing without velocities—only two other systems have been shown to solve this: CE and ESP…
NEAT (the name of their system) takes 25 fewer evaluations than CE and is five times faster then ESP.

FUTURE WORK—NEAT handles complexification relatively well—meaning that it can incrementally gain new functionalities without losing what it has learned.
Other directions including applying same genome innovation coding to things besides neural networks.