Evolving NNs through Augmenting Topologies

Background

Topology and Weight Evolving Artificial Neural Networks (TWEANN)

Neuroevolution searches through the space of behaviors to find a network that can optimize a solution. There are many difficulties: the search space is very large. How can we do this efficiently, how can we deal with high dimensional tasks that often require frequent tuning by a human observer? NeuroEvolution of Augmenting Topologies (NEAT) comes from Kenneth O. Stanley and Risto Miikkulainen at The University of Texas at Austin. NEAT puts forth a persuasive algorithm that addresses common GA issues. They claim that if done correctly, the evolution of structures and weights can be done more efficiently than existing TWEANNs.

Our report explores the advantages of NEAT, through ablation and comparison. First, we walk through what NEAT does and why it is better, then we ablate a key component of NEAT to validate its effectiveness, and lastly, we compare NEAT with Q-Learning.

The experiments used OpenAI Gym's CartPole-v0 along with the Python package neat-python, a NEAT implementation.

Genetic Encoding

TWEANNs use different schemes to encode genetic representations, of which there are two types: direct and indirect. Direct encoding schemes are literal representations of a phenotype. Indirect encoding schemes specify how a phenotype may be constructed.

1. Binary encodings are bit string representations of the connection matrix of a network. Simple, but limited.
    a. The size is proportional to the square of the nodes.
    b. Bit strings (and therefore number of nodes) must be the same size for all organisms. If a research chooses suboptimal size, experiment must be restarted with new size.
    c. It is difficult to compute useful crossover combinations with linear strings.
2. Graph encodings represent graphs more naturally. A dual representation scheme has a graph structure and a linear of node definitions to specify incoming and outgoing connections.
    a. Size grows similarly to binary encoding.
    b. Sub-graph swapping can now be done, but can't be sure if optimal swap.
3. Nonmating uses graph encoding, but without crossover. GeNeralized Acquisition of Recurrent Links (GNARL) shows crossover is not needed in TWEANN..
4. Indirect encoding. Cellular Encoding (CE) genomes are now programs written in a graph transformation language. CE focuses on cell division which spawn different types of

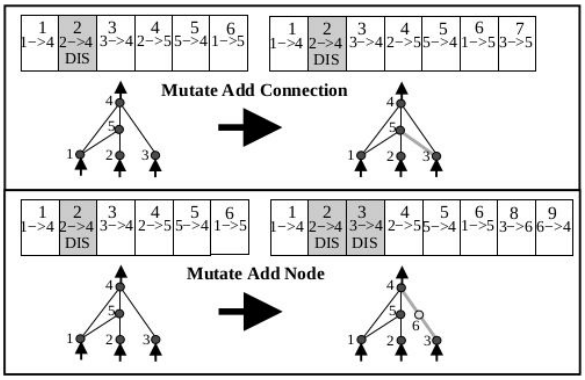
connectivities. CE genomes are compact and more closely represent the true nature of organisms.

a. However, we “requires more detailed knowledge of genetic and neural mechanisms”. This means encodings are not one-to-one with their phenotypes and may unpredictably bias the search.

Because of reason 4a, NEAT chooses direct encoding with a list of connection genes referring to two nodes being connected, along with the weight, state (enabled, disabled), and innovation number (more later). Node genes are a list of input, hidden, and output nodes. Structural mutations occur through add or delete node in a way that does not drastically alter the initial behavior of the network after mutation.

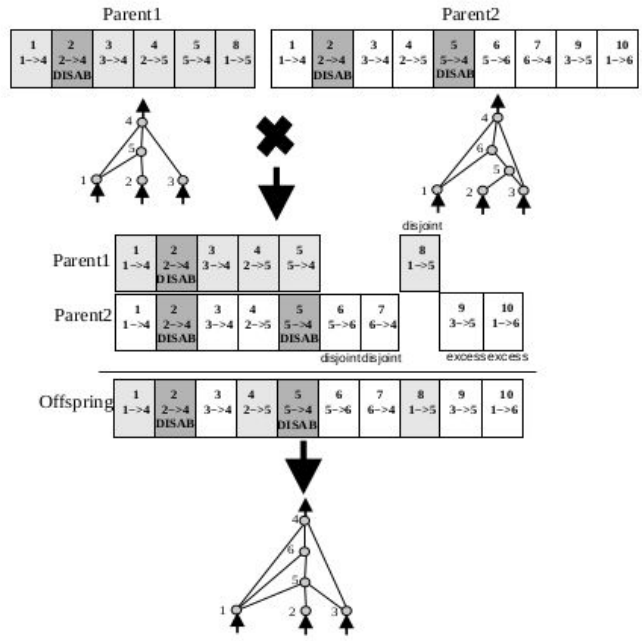
Tracking Genes through Historical Markings

Being that there are many ways to arrive at a solution with different networks, there arises the *Competing Conventions* problem. This problem occurs when two optimal networks crossover, a part of the solution may be lost in the resulting offspring. NEAT uses historical markings in its genes to supply evidence of homology. This was, new structure may be added without losing out on good genes.



Global innovation number keeps track of genes

Historical markers are placed by a global increment in genes where mutations occur. Matching genes are based on historical markers, whereas disjoint or excess genes are not. Now the system can form diverse populations through sensible mutation and mating.



Historical markers show when mutations occured.

Protecting Innovation through Speciation

Often, adding new structure hurts the fitness of a genome. If no time is given to develop this potentially optimal network, it may be cut. Speciation uses a compatibility function to place similar genomes in the same species where they all share fitness payoff, *explicit fitness sharing*. Explicit fitness sharing ensures networks have a chance to mature.

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W}.$$

δ is a distance measure of the adjustable coefficients, *c*, with number of *E* excess, *D* disjoint, and the average weight difference *W. If δ is less than some threshold, the two genomes are of the same species.*

Species reproduce by first eliminating the lowest performing members, then the offspring replace the generation. This protects topological innovation.

Minimize Dimensionality

TWEANNs typically initialize with random populations and topologies. NEAT instead initializes minimally, thereby decreasing the search space and minimizing dimensionality. This gives NEAT a good perfomance advantage.

Ablation

To validate NEAT, the paper ablates certain aspects of the algorithm to demonstrate the nessecity of each component. In the validation, the most significant component was the growth factor. Since the config file of neat-python allowed us to easily limit the growth, we decided to try the experiment ourselves.

Method	Evaluations	Failure Rate
No-Growth NEAT (Fixed-Topologies)	**30,239**	80%
Nonspeciated NEAT	**25,600**	25%
Initial Random NEAT	**23,033**	5%
Nonmating NEAT	**5,557**	0
Full NEAT	**3,600**	0

NEAT ablation summary from paper

No-Growth

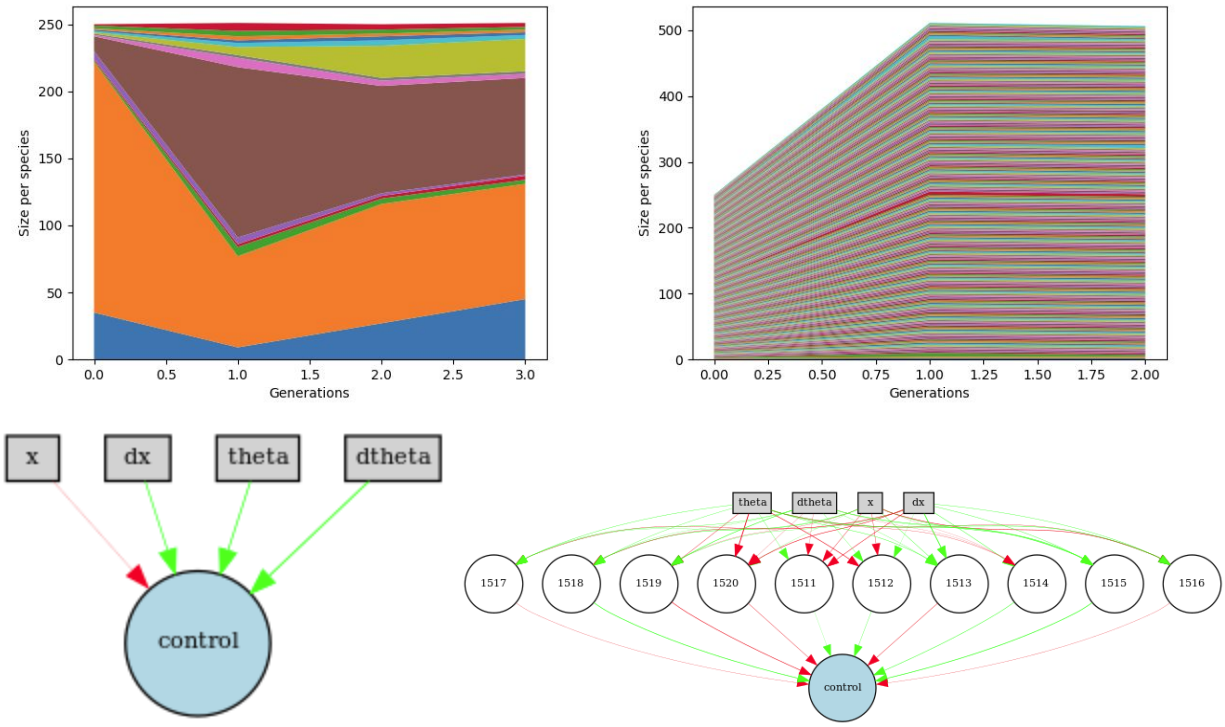
Disabling growth in a system would essentially be disabling the NE aspect of NEAT. Here we explore the effects of disabling growth. On the cart pole, reward was given for every timestep the pole was kept upright while the cart maintained velocity under the default cart pole velocity parameter for 20s (1000 time steps). We set 10 fully connected hiddens with no probability to add or delete connections or nodes, all else being equal. The following neat-python config options used:

```
num_hidden = 10
initial_connection = full_nodirect
conn_add_prob = 0.0
conn_delet_prob = 0.0
node_add_prob = 0.0
node_delete_prob = 0.0
```

Due to limited resources, this test was run 10 times normally and 10 times with no growth. The failure rate for regular NEAT was about 50% while non-growth NEAT was 80% and on average the non-growth had lower fitness scores. This is consistent with the results above. If we were to further tweak the coefficients and other parameters we likely would have seen the regular NEAT algorithm hit a lower failure rate.

	fitness									totals
NEAT	29	56	821	661	1000	1000	1000	1000	763	1000	7330
nogrowth	228	52	41	226	1000	1000	96	618	49	34	3344

Ablation results, when nogrowth contraint added, the system suffered



Left, NEAT. Right, non-growth NEAT, both successful.

When given no chance to grow, the algorithm saw each network as a separate species. Since the campatibility function groups species based on historical markers in structures, the constant state structure was fully separated, ie. no individual network was in the same species as another network. So each network was still able to take advantage of speciation since they didn't have to compete with other species that may overpower.

Q-Learning Results

During testing and comparison there were a number of noticeable results. The first of which is that when we first began experimenting with Q-Table results using both the Mountain Car and Cart Pole OpenAI environments that the state space became truly massive. This wasn't entirely unexpected.

In an attempt to counteract the explosion of observation space, we implemented a simple rounding method and applied it to observation numbers that are returned by OpenAI's API. By default, OpenAI returns a numpy array of some size that corresponds to the agent's observation of a given environment. In our two environments the observations consist of a 2x1 array for the Mountain Car, and a 4x1 array for the Cart Pole.

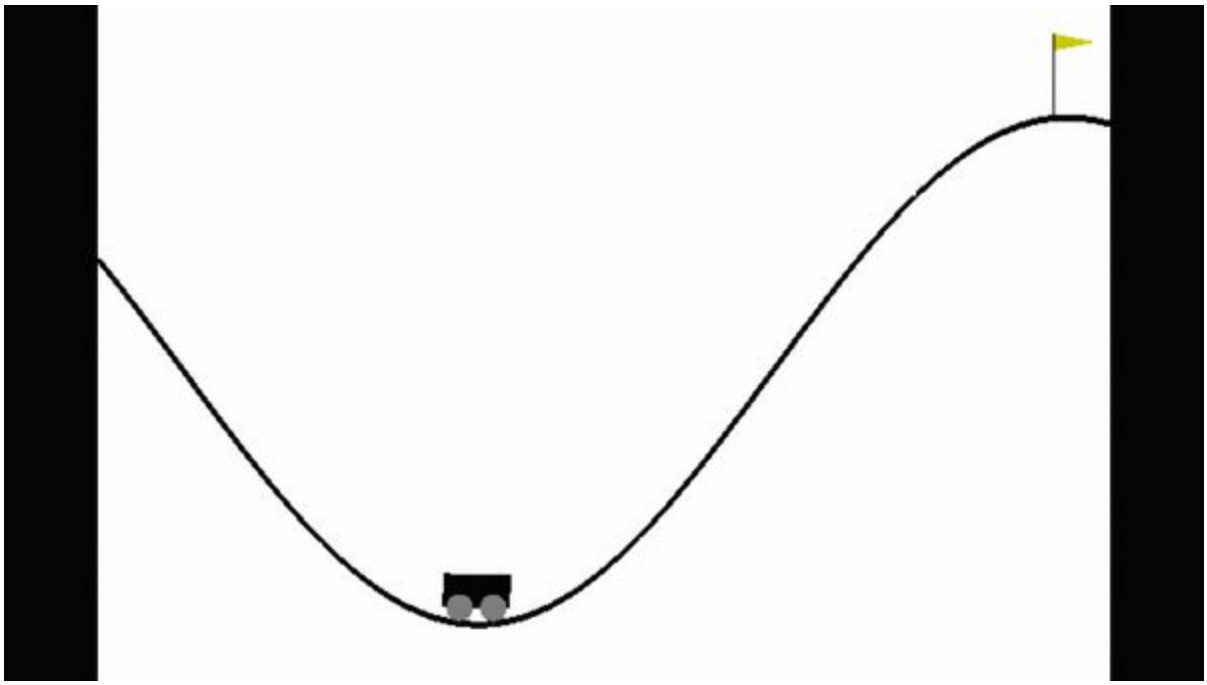
Normally these array sizes would be trivial to handle. However, the observations can return floating point numbers, so there is a state-space explosion happening here that, if not rounded, makes Q-Learning with a table simply not possible. Regardless, we made an attempt

to see if there was potential in the idea and if there would be any result that we could draw inference from.

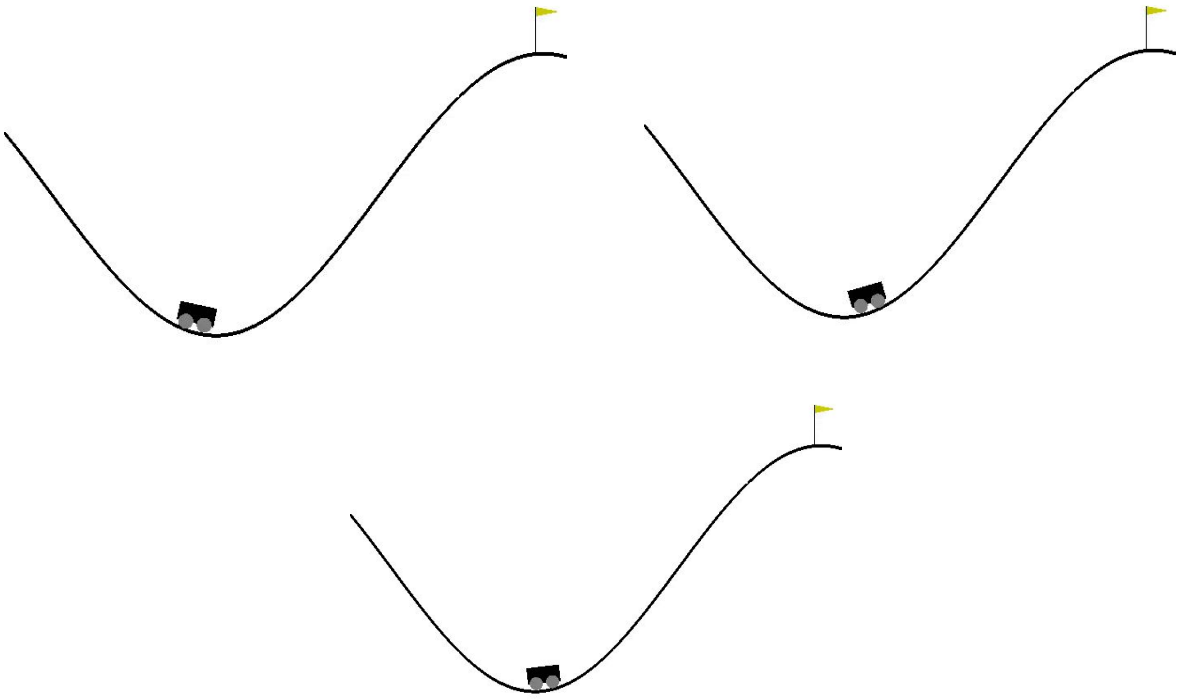
When we picked the OpenAI environments for testing, we had to be cautious as using something like one of the atari game environments would not be possible to make a meaningful comparison with. The documentation stated it was RGB 28x28 for inputs, and it doesn't take much convincing to see how the basic Q-Table method simply wouldn't do well.

Mountain Car

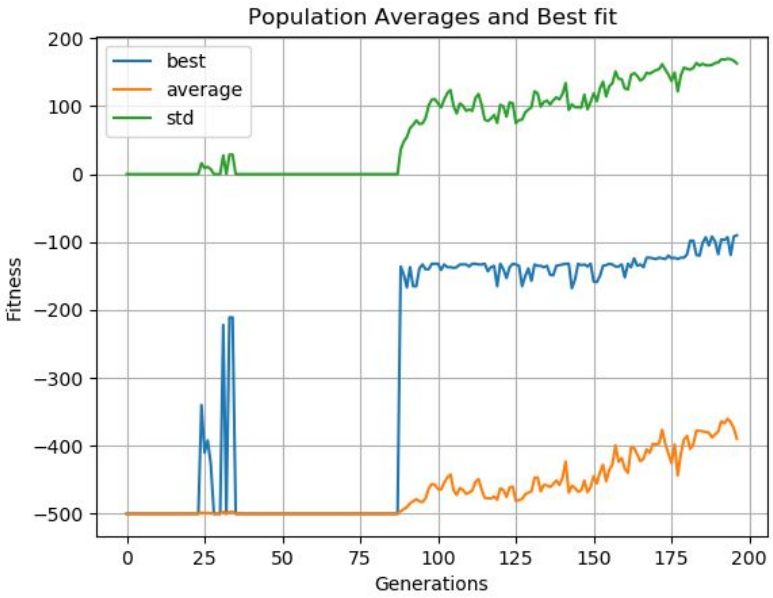
The mountain car environment is the most interesting of the two environments. Because our implementation of the Q-Table is highly stochastic at first, it produces erratic and exciting movement almost immediately. For every frame of action that the game runs, the agent receives a -1 reward. Here is an example of the Q-Table's behavior:

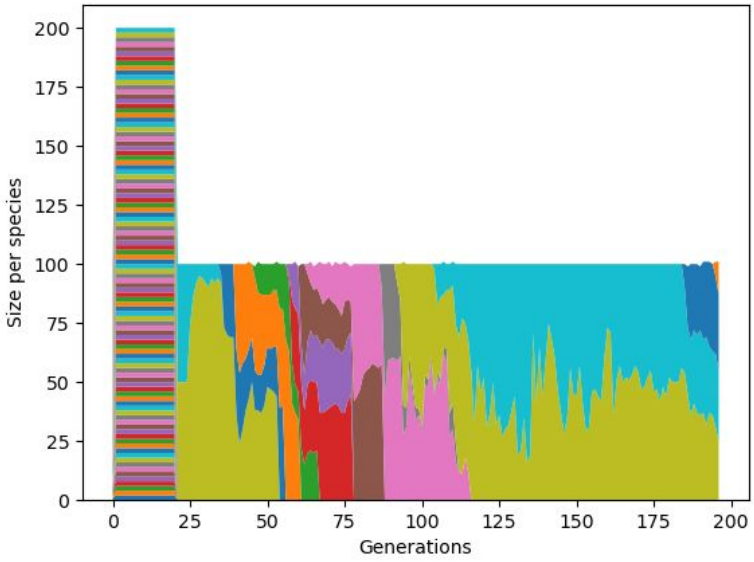


What makes this behavior exciting is that it was achieved with just a few hundred epochs. We believe that if the reward were based on the y-positioning of the agent rather than the current negative reinforcement of the environment this problem would be incredibly simple. We decided that, in order to preserve the integrity of the comparisons, no changes should be made. However, with tweaks to the Q-Table algorithm we believe this could show promise in an environment that is challenging for the agent. Here are the accomplishments made by NEAT:



It isn't explicitly shown on the gif images themselves, but finding a result took about 20-30 minutes. Moreover, we had to use a population size of 100 with 4 hidden layers over 1000 generations to do so. What makes this result possible is the 'soft reset' nature of the genetic algorithm that is incorporated into NEAT. The 'reward' for this agent is completing the track and avoiding penalty, so until that point reinforcement learning has no basis for a solution-bound action sequence. Therefore, this problem is heavily reliant on stochastic behavior by the agent, so random restarts is one of the most effective ways to approach it. Otherwise it's computationally expensive since the agent may be exploring greedily in some cases.





We can observe in these graphs what is happening inside of the NEAT program. As is atypical of genetic algorithm behavior, there is a long period of little to no solution progress until we 'win the genetic lottery' and there is a sharp increase in performance. Gradually, the highest performing agent breeds with the lower performant agents, producing a gradual increase in mean performance across all of the networks.

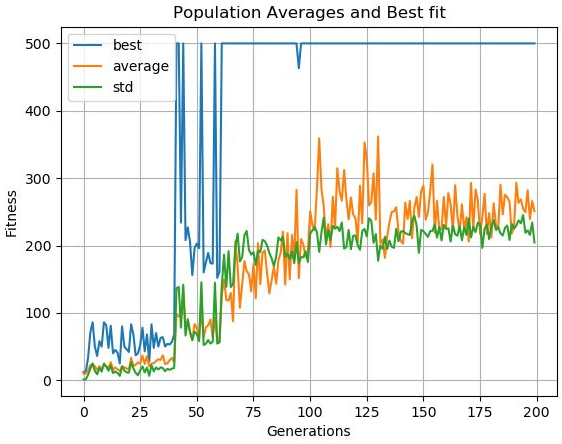
That evidence is further reinforced by the generations to species graph. In the first 20 generations there is a large species diversity, but as NEAT runs it begins to prune poor performing agents in an event called *extinction*. Finally, we're left with a few species that are producing results and those are the colors we see at the end of the generational graph.

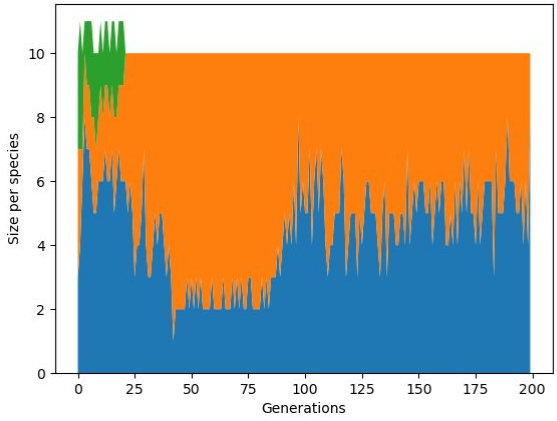
Cart Pole

Moving on to the Cart Pole example using a Q-Table, our results were much less exciting and we ended up with an agent that shows some attempt at a solution, but otherwise it's not quite able to learn the environment.

As can be seen, the agent can briefly hold the pole upright but fails quickly. The Q-Table trained agent repeats this behavior for the remainder of our tests, and if the observation space is rounded to the nearest 10th of a digit we can reach state spaces as large as 4000 or more.

Using the NEAT program, we were capable of reaching pleasant results in a sparse number of generations. As few as 50 with no hidden layers, and a population size of 20. A video link to our results: https://www.youtube.com/watch?v=FuBSOWEh910&feature=youtu.be.





These graphs tell a story similar to what we've documented in the mountain car results, but the fitness is more characteristic of reinforcement learning. The average population agent underperforms, but with as few as 50 generations it's evident that a perfect performing agent is achievable. In this environment, the reward is +1 for each frame that the pole is held up, so the top of the graph indicates a learner that has mastered the game because there are only 500 available steps here, so an agent's maximum reward is 500.

Sources

NEAT:

http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

Environments used for experiments::

https://gym.openai.com/

NEAT Python library:

https://github.com/CodeReclaimers/neat-python