Comparison of Neural Evolution of Augmenting Topologies and Deep Q-Learning

Tyler Petty
Computer Science Graduate
Washington State University
Vancouver, United States of America
tyler.petty@wsu.edu

***Abstract*—Neural Evolution of Augmenting Topologies and Deep Q-Learning are powerful reinforcement learning algorithms that solve a given problem in very distinct ways. Three experiments utilizing the cart-pole problem were conducted in order to determine which of these two algorithms are more accurate, and have the shortest training and running times.**

I. Introduction

A. Neural Evolution of Augmenting Topologies

Neural Evolution of Augmenting Topologies, commonly abbreviated as NEAT, is a reinforcement learning neural network that uses principles of genetic algorithms to create a minimal sized neural network for solving a given problem. This is done in a way similar to natural selection which leads to the components of the algorithm being named after biological terms. The general structure of each network, or genome is shown in figure 1 which shows that each genome consists of a list of nodes and connections. Each node consists of a node id, or number, and node type while connections are described by an input node, output node, weight, whether or not the connection is active, and an innovation number. The innovation number is a critical component of each genome as it allows for the genomes to crossover, creating more genomes with similar aspects to both parents.

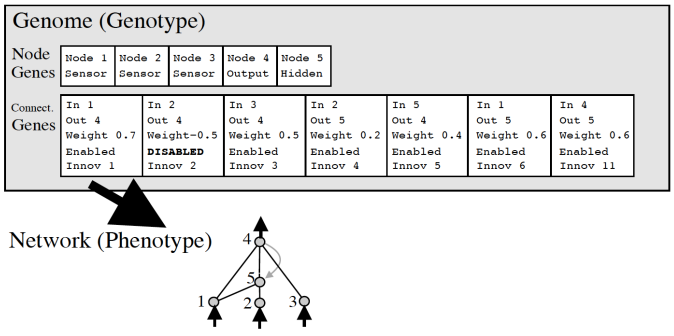

Fig. 1. How each network is structured [1]

NEAT works by creating a population of genomes and runs the population on the problem, evaluating each genome on its performance using a fitness function. After the population has finished, it separates the genomes into groups called species based on the nodes and connections that describe the genome. Figure 2 shows how these species were made across 5 generations. The algorithm started with every genome in the same difference because there was negligible difference between them, and later spit up into 3 different species.

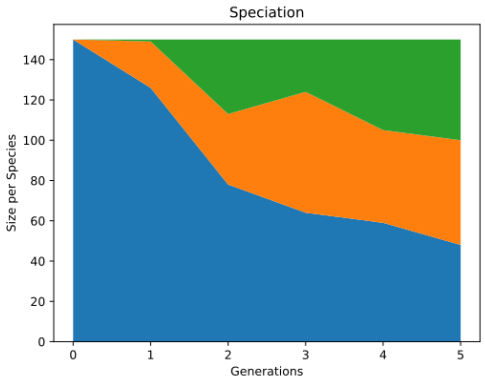

Fig. 2. Speciation of NEAT for 5 generations

Once the species are filled, many of the genomes are removed, keeping only a selection of the best performing genomes. At this point, the population is filled with a combination of breeding genomes or mutating the best performing genomes from each species. The act of breeding two genomes is shown in figure 3. This shows that two genomes of differing topologies can crossover to create a new genome by using the innovation numbers stored in the connection genes to create a sorted list of each gene, allowing for blank genes to fill in the gaps. During crossover, for similar genes, the new network will get the gene of the best performing parent while receiving direct copies of disjointed genes. Mutations are defined as any operation that changes the structure or weights of a genome. The algorithm uses various types of mutations, including adjusting the weights of existing connections, creating new connections, adding or removing nodes, and enabling or disabling connections. A mutation can also change what activation function the genome uses or what type of calculation each node will perform to generate an output. Crossovers and mutations across multiple generations allow NEAT to create a minimal sized neural network that will solve the given problem.

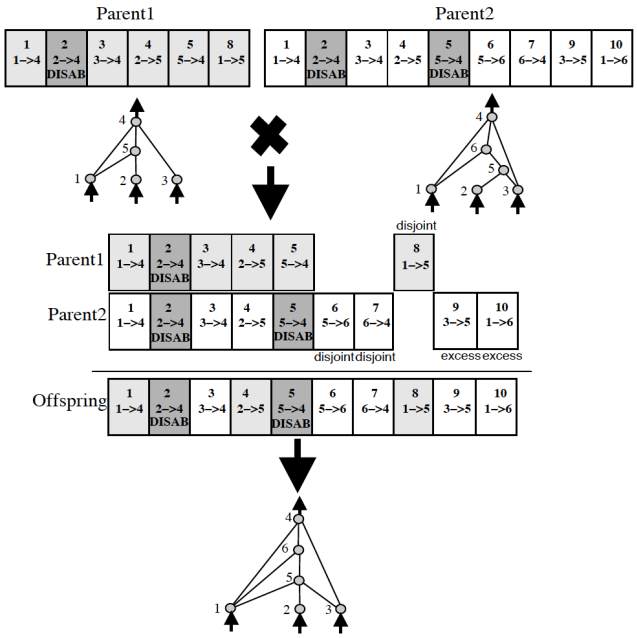

Fig. 3. How new networks are created using two existing networks [1]

B. Deep Q-Learning

Deep Q-Learning (DQN) neural networks are one of the most frequently used reinforcement learning algorithms and are based on the Q-Learning or Q-Table algorithm. Unlike the formerly stated algorithm, DQN evaluates the network after each action, assigning a reward or q-value for an action performed at a given state. This q-value is calculated with equation

$$Q^{new}(S_t, a_t) \leftarrow (1-\alpha)*Q(s_t, a_t)+\alpha*(r_t+\gamma*maxQ(S_t, a) \quad (1)$$

with an as the action, α as the learning rate, γ, r as the reward and s as the state. As shown with the $Q(S_t, a_t)$, every time this equation is computed, the action and state are stored in a table so that the algorithm can reference this later. One of the downsides of this is that for many non-trivial problems this table can get quite large, this is where the neural network comes in. A neural network can be used to replace the need for a large table by learning from previous actions and create a model that can predict the best action to perform at a given state. This allows for the equation to be reduced to just the future reward

$$Q^{new} \leftarrow r_t + \gamma * maxQ(S_t, a) \quad (2)$$

While the addition of a neural network removes the need for a large Q-Table by predicting the correct action for a given state. This network is trained uses a randomly selected subsection of the q-table every episode. Figure 4 shows an example of a run of DQN on the cart-pole problem used for the following experiments. As shown, through most of the run DQN is unstable and would rarely behave similarly to a previous run until later in the run when it finally stabilized towards a

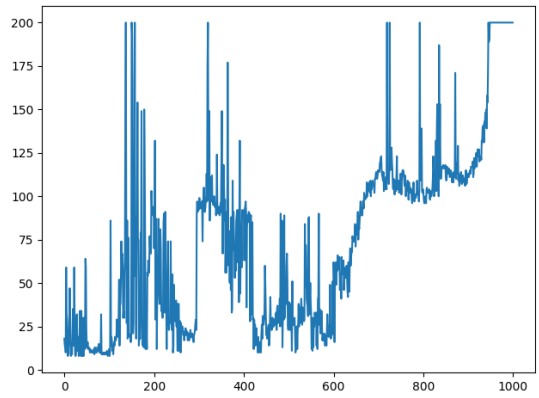

Fig. 4. Rewards of each episode of training

optimal neural network. This is due to DQN exploring random states at a high rate at the beginning of the run and gradually decreasing on exploration as the algorithm runs. This run was able to solve the cart-pole problem with an average score of 200 over 100 consecutive runs.

II. Experimentation

For each experiment the cart-pole problem, described in "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem" [2] and implemented by OpenAI [3], will be used to evaluate each reinforcement learning network on three criteria. The cart-pole problem provides each network the state of the environment consisting of the cart position, velocity, the angle of the pole and the velocity of the tip of the pole. The networks will take this state and predict the appropriate action, either move the cart left or right. Each run of the cart-pole problem is ranked by how many ticks, or different states, the network can achieve without reaching a failing condition up to a maximum of 200. These failure conditions are if the cart reaches the edge of the environment or the pole exceeds an angle of 12.8 degrees from center. In order for the problem to be considered solved, a network will need to have an average score of 195 or greater over 100 consecutive runs.

The criteria used to evaluate each network are how long each network takes to train, how long they take to execute 100 runs, and the average score of those 100 runs. The python library Neat-Python [4] is used to implement the NEAT network while the library Keras [5] will be used for the neural network in DQN. Each network will be executed 5 times for each experiment in order to get an average performance.

A. Experiment 1: Training until first success

The purpose of this experiment was to determine how quickly each reinforcement learning algorithm can find a solution for the cartpole problem. Once a network achieved a score of 200 it was tested on 100 consecutive runs of the cart pole problem and the following data in table I. With this experiment, the NEAT algorithm performed better than the DQN in every aspect, with an average score 40% higher

TABLE I
STATISTICS FOR TRAINING UNTIL AGENT REACHES SCORE OF 200

Run	Results for NEAT		Time	
	Average score	*Generations*	*Training*	*Testing*
1	194.7	5	0.186887264	0.347672224
2	180.46	3	0.076301575	0.270162821
3	148.95	4	0.188971758	0.209892511
4	148.04	5	0.222026587	0.226582289
5	124.31	3	0.180119276	0.229402781
AVG	159.292	4	0.170861292	0.256742525
Run	**Results for DQN**		Time	
	Average score	*Episodes*	*Training*	*Testing*
1	161.54	192	42.89829373	5.936439037
2	79.22	153	14.77516103	3.044193029
3	193.09	206	23.47548318	7.420827866
4	15.62	1000	62.88223815	0.664268017
5	116.86	204	29.43828607	4.253621101
AVG	113.266	351	34.69389243	4.26386981

achieved within 1% of the number of iterations required by DQN. The run times for NEAT where also far shorter than DQN with training only requiring less than 0.5% of DQN and testing 0.6%.

B. Experiment 2: Training for 1000 iterations

The second experiment shows how likely the final model is for each algorithm will solve the cart-pole problem. It also shows the time required for them to complete the same number of iterations, which proved to be a better baseline to compare the timing of each. The data from this experiment is shown in table II

TABLE II
STATISTICS FOR TRAINING 1000 EPISODES OR GENERATIONS

Run	Results for NEAT	Time	
	Average score	*Training*	*Testing*
1	200	298.1772931	0.300250053
2	176.87	301.5972867	0.318750858
3	199.51	334.8321354	0.320248365
4	199.96	301.4146879	0.334125996
5	182.38	354.7409582	0.302638769
AVG	191.744	318.1524723	0.315202808
Run	**Results for DQN**	Time	
	Average score	*Training*	*Testing*
1	169.08	492.12236	6.259386063
2	197.64	530.5806079	7.878951073
3	181.14	388.883491	6.740566969
4	200	280.2255721	8.820346832
5	200	527.3328021	7.466546059
AVG	189.572	443.8289666	7.433159399

The results of this experiment are very similar to the first, with NEAT still acceding in all criteria, however the average scores and training times are a lot closer than they were previously. Comparing the average scores from the two experiments shows that longer training times results in a better model, likely because it allows the networks to train on more states of the environment and tune in towards the optimal solution.

C. Experiment 3: Training until completion of problem

Experiment 3 continues training until a network is capable of achieving an average score greater than 195 for 100 runs. When a network achieves a reward of 200, it is tested on 100 runs and continues training if the average score doesn't exceed the threshold. This will cause the training times to be far greater than in the previous experiments. This shows that the resulting model is able to continuously succeed in solving the cart-pole problem. Table III shows the results of this.

TABLE III
STATISTICS FOR TRAINING UNTIL AN AVERAGE SCORE OF AT LEAST 195

Run	Results for NEAT		Time	
	Average score	*Generations*	*Training*	*Testing*
1	200	7	0.9424	0.3554
2	200	8	1.3277	0.3340
3	199.71	22	6.2844	0.3117
4	200	7	1.1217	0.3126
5	196.13	7	1.0898	0.3053
AVG	199.168	10.2	2.1532	0.3238
Run	**Results for DQN**		Time	
	Average score	*Episodes*	*Training*	*Testing*
1	200	380	1298.7858	7.4776
2	199.75	169	204.4820	7.44421
3	198.6	171	225.8169	7.4076
4	200	537	1515.3439	7.6976
5	198.87	178	186.8369	7.3826
AVG	199.444	287	686.2531447	7.481975985

As the results show, both NEAT and DQN were capable of completing the cart-pole problem with similar average scores, however like the previous experiments, NEAT continues to perform the experiment in less time than DQN.

III. CONCLUSION

NEAT is capable of running the cart-pole problem far faster and with fewer iterations than DQN with similar or better results. The faster run times are likely due to NEAT running the problem multiple times each iteration and keeping the smallest best performing genomes, starting from a network with only input and output nodes. This allows it to creating a minimal sized network through small alterations that is still capable of completing a given problem. In contrast, DQN requires a predefined neural network and is only capable of adjusting the weights of the connections, resulting in every iteration having the same topography. Another reason that DQN has longer run times is because after every iteration, it has to train the network on a selection of previously explored actions and states.

Deep Q-Learning and Neural Evolution of Augmenting Topologies are powerful algorithms for reinforcement learning, each with their strengths and weaknesses, however using a simple and well defined problem such as the cart-pole balancing problem as a benchmark, NEAT is capable of creating a model that outperforms what DQN can create in less time and with fewer iterations. This supports Stanley's statement that NEAT is capable of outperforming fixed topology networks such as DQN.

REFERENCES

[1] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[2] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, Sep. 1983.

[3] OpenAI, "Cartpole-v1," https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py, 2019.

[4] A. McIntyre, M. Kallada, C. G. Miguel, and C. F. da Silva, "neat-python," https://github.com/CodeReclaimers/neat-python.

[5] F. Chollet *et al.*, "Keras," https://keras.io, 2015.