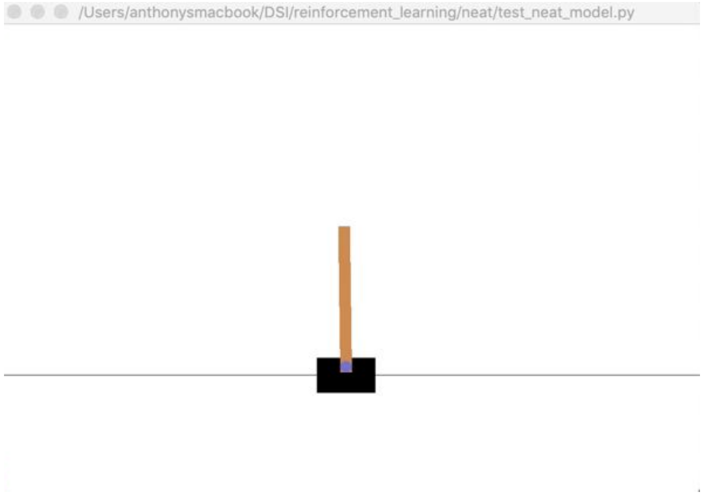
Reinforcement Learning

By Anthony Clemens

Abstract

- Used openAI gym to train reinforcement learning models to solve the cartpole control problem in simulation
- Implemented two solutions:
  - Deep Q-Network (DQN) algorithm
  - Neuroevolution of Augmenting Topologies (NEAT) genetic algorithm
- Shown these methods are capable of solving a classic control problem



Problem Statement

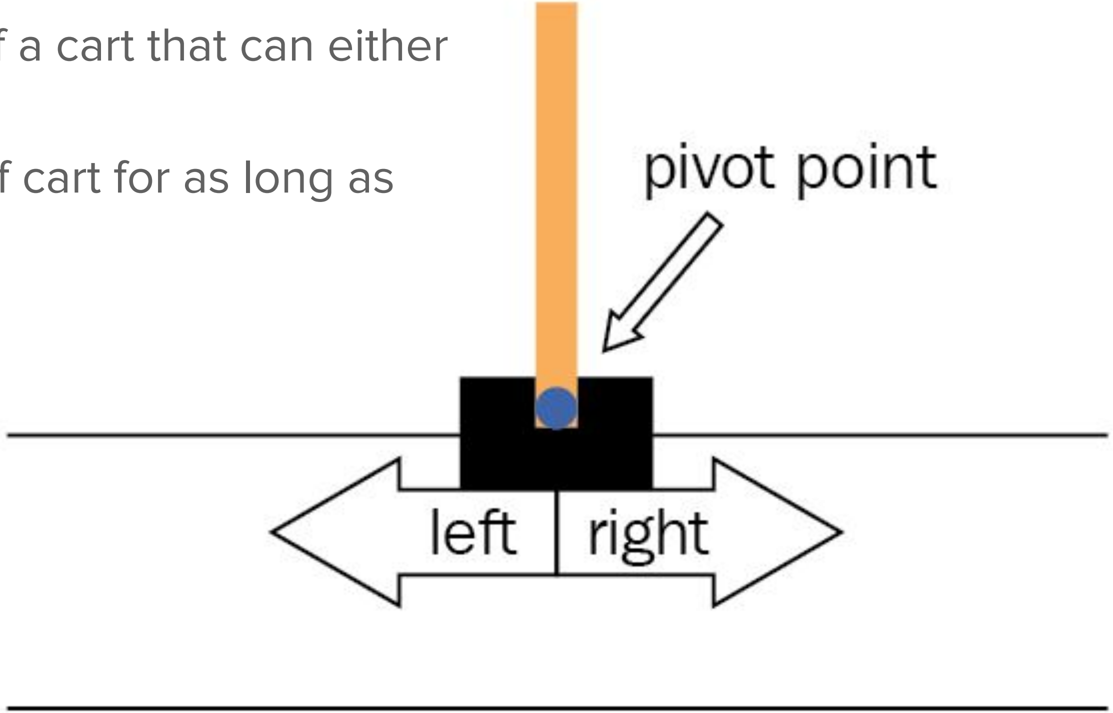
Are modern reinforcement learning and genetic algorithms such as DQN and NEAT capable of solving a classic control problem in simulation without being explicitly told how to behave?

Why do we care?

- The applications of training models in simulation are tremendous
  - E.g. Self-driving cars, robotics, live stock market trading, poker bots
- Training in simulation saves time and money (in simulation car accidents don't matter)
- Future: train models in simulation, and bring them into the real world

Cartpole Problem

- A pole is balanced on top of a cart that can either move left or right.
- Goal: balance pole on top of cart for as long as possible
- State:
  - Cart position
  - Cart velocity
  - Pole angle
  - Pole velocity
- Action:
  - Move left
  - Move right



Q-Learning

Based upon the idea of a Q-function:

The Q-function quantifies the reward an agent may receive given its current state and next action.

Note that depending upon the complexity of the problem, it may be too difficult to write the explicit mathematical formula for the Q-function by hand.

Deep Q-Network (DQN)

- Neural network is used to approximate the Q-function

$\therefore$ **The explicit mathematical formula for the Q-function is not needed**

- The neural network maps the current state the agent is in to the next action with greatest reward.

The Exploration Rate

- We first train a model using random actions, and then slowly start trusting our model's policy (strategy)

The exploration rate is the parameter that quantifies whether we choose a random action, or the model's action

- The exploration rate is initialized at 1 (pure random actions) and decays (uses the model more) as training proceeds.
- The exploration/exploitation trade off is the dilemma of trusting our model (exploitation) and getting an expected result or choosing a random action (exploration) and possibly learning something new.

Neuroevolution of Augmenting Topologies (NEAT) Genetic Algorithm

- Novel algorithm, developed by Ken Stanley in 2002
- Not only alters the weighting parameters of the neural network, but also the architecture of the neural network itself.
- The beauty of the NEAT algorithm is you don’t need to know the best neural network architecture for the problem at hand

∴ The algorithm will produce the best neural network architecture.

How does the NEAT genetic algorithm work?

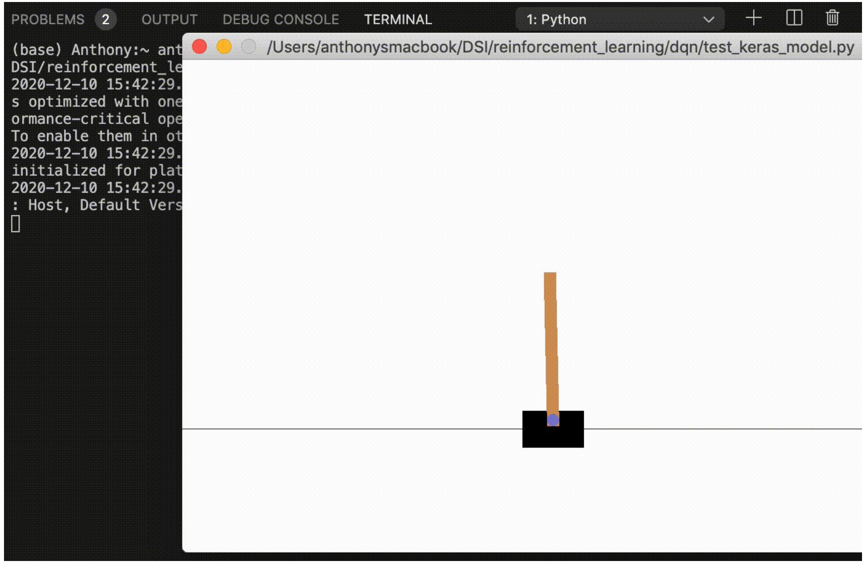
- Starts with a population of genomes. Each genome contains two sets of genes that describe how to construct a neural network:
  1. Neuron genes, specifies a neuron
  2. Synapse genes, specifies the connection between two neurons
- A fitness function quantifies the quality of an individual genome.
- Fitness function is used to evaluate each of the genomes

How does the NEAT genetic algorithm work? Cont.

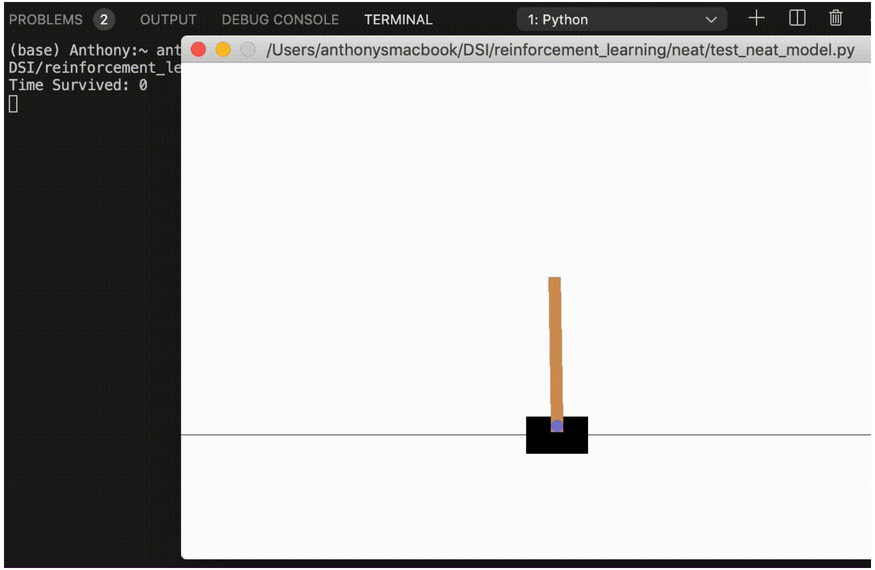
- The next generation is produced through reproduction and mutation of the fittest individuals.
- The reproduction and mutation operations in the algorithm may add neurons and/or synapses to genomes
- As the algorithm proceeds genomes will increase in complexity.

Results

DQN



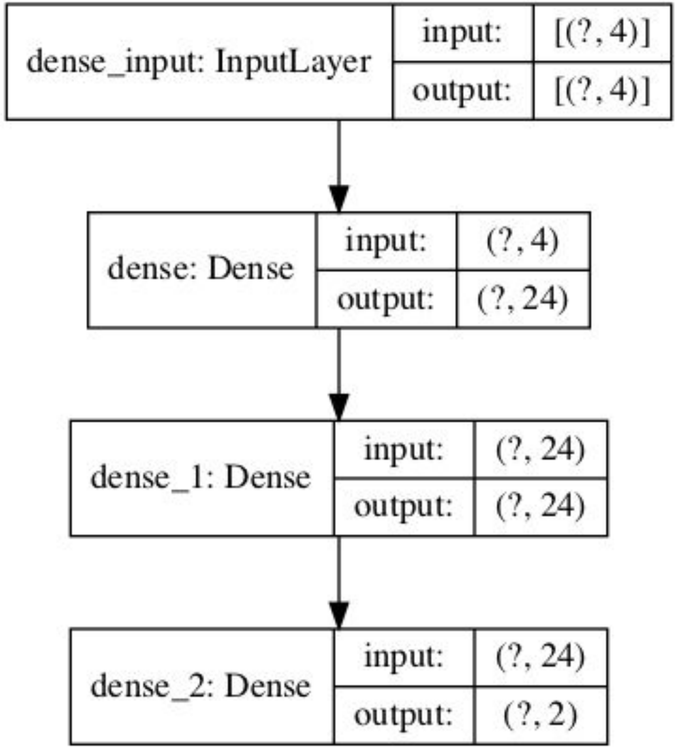
NEAT



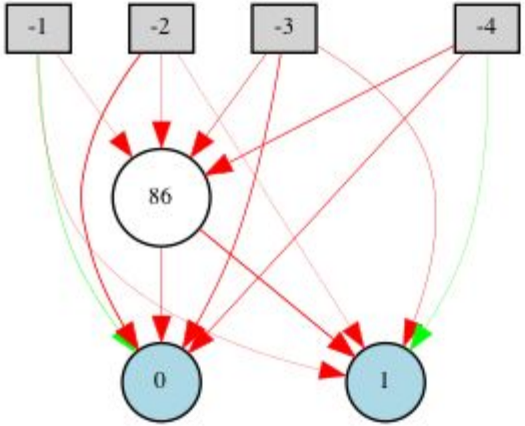
- Both methods were able to obtain a solution.
- NEAT is faster

Results: Neural Networks

Keras:



NEAT Fittest Genome:



Conclusion

- Both methods were able to construct a successful policy without being explicitly told how to behave.
- I believe NEAT is superior in this context
  - Simpler solution
- We were successful in training models in simulation.
- The applications of training models in simulation are tremendous

Questions?