Introduction to Neuro-Evolution

Andrew Geng, Jonas Klare, John Balis

December 13, 2018

1 Abstract

In this activity, we will build up to neuro-evolution by looking at the essential components, culminating in a activity showcasing the ability of neuro-evolution to create intelligent agents. The first objective is to learn about the potential value of nondeterminism. This forms the basis for genetic algorithms which in turn form the basis for neuro-evolution. For our first activity, we illustrate the differences between simple gradient descent and simulated annealing to provide a motivation for the use of nondeterminism.By the end of this sections, learners should understand simulated annealing, and why nondeterminism is sometimes useful in learning algorithm design. From here, we present a simple genetic algorithm, training on the same problem. By the end of this excercise, learners should feel comfortable with the concepts of mutation and crossover, as they apply to genetic programming, and have a general understanding of how to structure a genetic algorithm. In the final section, we highlight some of the pitfalls of the neuro-evolution process, including training time and reliability, while also showcasing the potential of genetic algorithms to create intelligent agents. By the end of this section, learners should have an appreciate for both the difficulties and potential power of genetic algorithms!

2 Basics of Mutations

Background
In problem 1 we will showcase a comparison between traditional Gradient Descent algorithm and a mutation based algorithm. For simplicity of visualization, weve created a simulation of these algorithms using a traditional hill climbing approach and a simulated annealing approach.

Simulated Annealing is a probabilistic machine learning technique used for solving for the global maximum/minimum for a provided function or search space. Inspiration for the technique is derived from annealing from metallurgy, thus its

given name simulated annealing [2]. We used this methodology due to its classification as meta-heuristics and due to its functional similarities with Genetic Algorithms. Critically, we will use the non-deterministic properties, defined within a traditional simulated annealing algorithm, to showcase what a mutation within genetic algorithms seeks to achieve.

The function used to calculate the acceptance probability is shown below [1].

$\text{P(Acceptance)} = (1 \text{ if } \delta c \leq 0, e^{-\frac{\delta c}{t}} \text{ if } \delta c > 0)$

Please note that you are not required to understand the following formula or have any previous knowledge regarding simulated annealing or hill climbing, we are simply giving you additional information if you wish to explore further. Additionally, note that links have been provided both in the source code and the work cited if you wish to learn more about each methodology.

Download the given mutation.m code and run mutation.m.

2.1

Figure 1 showcases a gradient descent algorithm with 50 different starting locations on the graph. The bottom graph showcases how many points are stacked ontop of the corresponding location in the top graph. Do all 50 of the points end up in the global minimum. If not why do they not reach the global minimum.

2.2

Figure 2 showcases a mutation based algorithm with 50 different starting locations on the graph. The bottom graph showcases how many points are stacked ontop of the corresponding location in the top graph. Do more of the 50 points end up in the global minimum. If so, why do you think they reach the global minimum more often.

2.3

Explain why a mutation (simulated annealing) algorithm may outperform a simple gradient descent algorithm?

3 Basics of Genetic Algorithms

Background

The motivation for this rests in nature where individuals evolve based on reproduction and a small chance for mutation. Our program attempts to mimic that. Each thing can be modeled as a genome that stores variables that represent certain properties. For example in our genome is described by (SPEED OF DESCENT , CHANCE OF JUMP,JUMP DISTANCE)

3.1

Run the program in PROBLEM_2_STARTER.m. To mimic reproduction, we will use sexual reproduction between two parents that will mix their genome to create children. After the program has terminated, look at the PARENT and CHILDREN variables. How do they compare? What would a reasonable child be given the parents (1,.5,3), (2,.1,5)?

3.2

With a small chance we will mutate. When the program has terminated, look at the variable MUTANT_CHILDREN. How do these compare to the last children and parents?

3.3

Note the avg fitness scores throughout generations. This is due to only the top performers being selected each generation from a fitness score. For our fitness function, we define the individuals that can find the lowest minima on a graph the most fit.

Look at the first graph. The 1st graph is where all of the parents land on the graph. Compare this to the 2nd graph where children that are run after 50 generations of breeding. Do the children perform better than the parents?

3.4

Did the genetic algorithm seem to perform better than the random initial parents? Please note the pros and cons of a genetic approach, considering the complexity of a problem and its correctness.

4 Neuro-Evolution

Background
Historically, it has been less popular to train neural networks by mutation, as

it can take a lot of computing power to arrive at good solutions this way. With the increase in available computing power, It has become more feasible to train neural networks via natural selection, preserving genotypes that result in good performance on some task, and throwing out genotypes that perform poorly.

The algorithm we demonstrate in this activity works as follows:

1.Create a random starting network with very few nodes and connections.
2.Test it to see how well it performs.
3.Create a mutated copy.
4.Test the copy to determine its score.
5.Replace the original with the copy if the copy gets a higher score.
6.Repeat this process

In this section, we will use the example of an agent trapped in a grid with a nondeterministic hunter as an example of how a naive genetic algorithm using neuro-evolution can learn intelligent behaviors. The 'world' is be displayed as a matrix with the hunter represented as a 7 and the agent represented as a value between 1 and 4 inclusive. This number indicates the direction the agent is facing: 1 being to the right, 2 being down, and so on. At each frame of time, the agent is given only the distance to the nearest object or wall in front of it, and whether or not the hunter is currently 'attacking' it. The hunter always advances towards the player at each from of time with a probability corresponding to the agent can move in it's cardinal directions, turn, or strafe. Agents who avoid the hunter for the longest are selected for by the algorithm.

Unlike the previous example, the model being mutated in this case is a spiking neural network(SNN) rather than a set of gradient descent parameters. The structure of the SNN is not covered in this lesson, but knowledge of the SNN's structure is not assumed or necessary to understand the activity. The algorithm we are using is a vastly simplified version of the algorithm used in [3](We do not employ crossover or keep track of 'innovation' numbers)

4.1

Run avoidance.m with 100, 200, 500, and 1000 epochs. Note you will need manually terminate the program when you are done observing the visualization. What differences do you observe in the behavior of the agents with respect to the number of training epochs? Don't be alarmed if you don't see any particularly intelligent behaviors.

4.2

Run avoidance.m in readFromFile mode. Make sure to uncomment line 10 and comment line 13. Also make sure 'bestAdj.csv', 'bestW.csv', and 'best-

Thresh.csv' are in the same directory. The network specified by these files was trained until it reached an average performance of 950 frames of time survived. How does the behavior of this network compare to that of the networks you trained in part A?

4.3

Next we can modify some of the simulation parameters and observe the effects on learning as well as the pre-trained network. Set speed from .2 to .4. Repeat part A and B with speed at .8. Optionally try changing the board size as well. Is there a change in the behavior of the pre-trained network? Is there a change in the behavior of the networks you are training? Hypothesize about what your observations mean when it comes to designing training simulations for neuro-evolution. This question is meant to be open ended, but we provide our findings in the solution.

References

[1] Gnanachandran. Using simulated annealing to solve logic puzzles. 2016.

[2] Surendra Nahar, Sartaj Sahni, and Eugene Shragowitz. Simulated annealing and combinatorial optimization. pages 293–299, 1986.

[3] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, June 2002.