Genetic Algorithms

Alan Zheng

April 25, 2018

1 Introduction

Much of what we as humans imagine and create is inspired by nature, and machine learning is no exception. Neural networks are inspired by the neurons in a brain and extract information from a dataset. Developed by John Holland in 1975, the genetic algorithm (GA) is a search-based optimization technique based on the principles of genetics and natural selection. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve.

2 Why Genetic Algorithms?

GAs are good when:

- there are multiple local optima
- the objective function is not smooth (so derivative methods can not be applied)
- the number of parameters is very large
- the objective function is noisy or stochastic
- there is no information about the gradient of the error function
- you want to parallelize
- you're lazy and don't want to implement something complicated

A large number of parameters can be a problem for derivative based methods when you don't have the definition of the gradient. In this type of situation, you can find a not-terrible solution via GA and then improve on that with the derivative based method. The definition of "large" is growing all the time. Also, there is no need for backpropagation, so GAs can be a competitive alternative for training deep neural networks for reinforcement learning for optimization problems.

3 Encoding Technique

In a GA, we represent each possible solution to the optimization problem as an individual chromosome. Information is stored inside a chromosome, which can be represented by a component vector, a vector, a string, etc. We call this the genome, or the genotype. In many situations, we can't tell anything about the individual or solution by just looking at the raw genome. We evaluate the genome using a fitness function to show the phenotype, to determine what it means or how well it performs in the context of the problem. We'll talk about this later. We choose the way we structure the chromosome depending on the problem we are trying to solve.

4 Initialization

To start off the GA, we generate a population, or a group of chromosomes. We may either use random generation or generation with a known heuristic for the problem. It's important to note that with heuristic generation, you shouldn't use the heuristic for the entire population, as it can result in the population having similar solutions and very little diversity. Upon initialization, we also use our fitness function to evaluate and score each of the population members. The fitness function looks at a chromosome (a solution) and gives it a number based on how good the solution is. It is important to note that the success of your algorithm highly depends on how well defined your fitness function is. Obviously, at this point, all the fitnesses will most likely be terrible.

5 Selection

Based off of the chromosomes' finesses, we select a constant number of them. There are several ways to do this:

5.1 Roulette Wheel

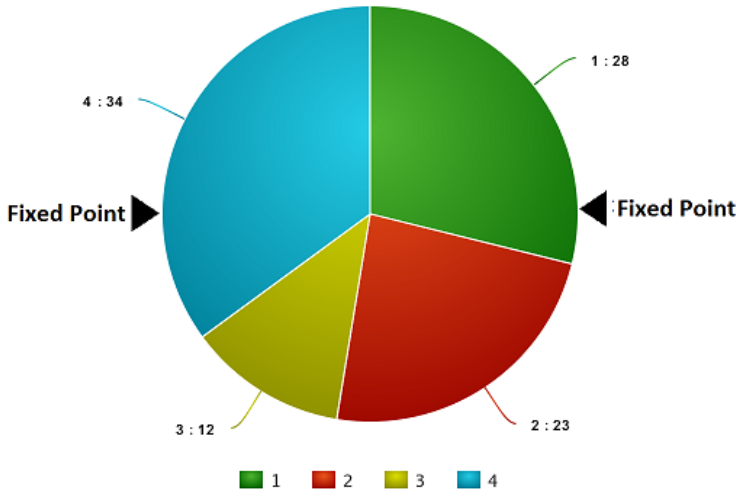
Using Roulette Wheel selection, we assign a probability to each chromosome which is directly proportional to its fitness.

$$P_i = \frac{f_i}{\sum_k^n f_k}$$

We then select a set amount of chromosomes based on their probabilities.

5.2 Stochastic Universal Sampling (SUS)

This is almost the exact same thing as roulette wheel selection, but with a little tweak. Where roulette wheel selection chooses several solutions from the population by repeated random sampling, SUS uses a single random value to sample all of the solutions by choosing them at evenly spaced intervals. This gives weaker members of the population (according to their fitness) a chance to be chosen and reduces the more unfair nature of fitness-proportional selection methods.



5.3 Rank Selection

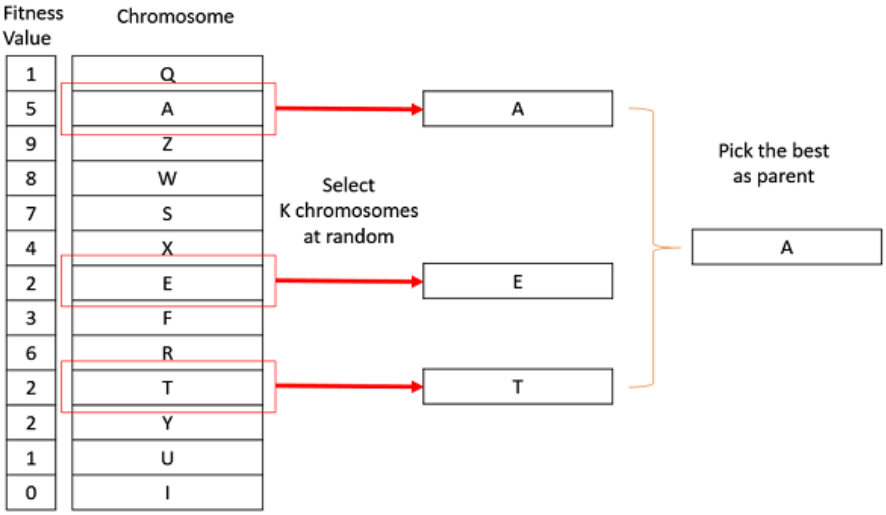
Using rank selection, we assign a probability to each chromosome which is directly proportional to its rank instead of its fitness. If there are n chromosomes, 1st place has rank n, 2nd has rank n-1, and so on.

$$P_i = \frac{r_i}{\sum_r^n r_k}$$

Like roulette wheel selection, we then select a set amount of chromosomes based on their probabilities.

5.4 Tournament

Using tournament selection, chromosomes are randomly sampled into multiple "tournaments," and the ones with the highest fitnesses within each group are selected.



5.5 Why?

The reason why we don't just select the chromosomes with the highest fitnesses from each population is that we don't want to get stuck at a local maximum. We allow lower fitness individuals to survive to explore other paths that could potentially be better. Here, we see hints of the exploitation vs exploration problem. For tournaments, if we want to lean towards exploration, increase the number of tournaments, and for exploitation, decrease it.

These selections are used to determine survival and which chromosomes to breed together in crossover. The selected chromosomes are allowed to continue while the rest are discarded from the population.

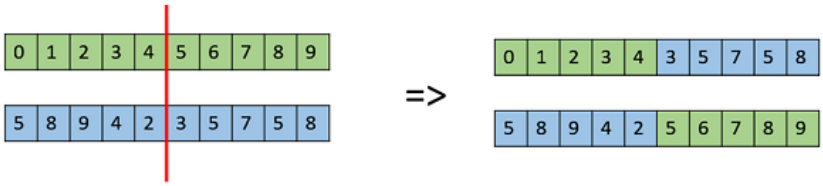
6 Crossover

The surviving selections are now candidates for parents! We need to crossover creating new children to replace the chromosomes we brutally murdered in selection. If we used roulette wheel selection, rank selection, or SUS, choosing parents is easy. All we need to do is use the saved probabilities from selection and use the stochastic universal selection method from SUS. Typically we choose 2 parents for each crossover, but you can do more. It is important to note that crossover is not a global search function. If you take a set of genetic vectors and perform various types of crossover on them, it will be folly to expect every conceivable vector from the crossover procedure. The reality will be that we are doomed to search within a finite space of possible vectors. To search effectively and beyond this finite space, you need something called mutation, which is effectively an injection of stochastic noise. However, crossover isn't entirely useless; it enables highly fit individuals to easily propagate their good genes across the population, so that improvements might be built, or found, on top of those.

Once we have the parents, different situations call for different operators for crossover. Some common ones include:

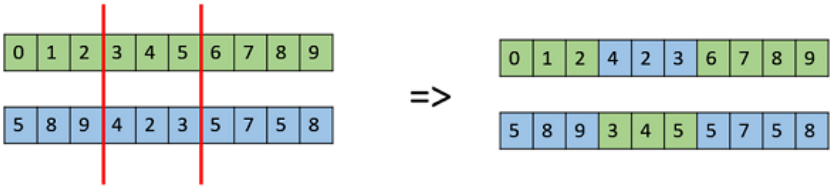
6.1 One Point Crossover

In one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.



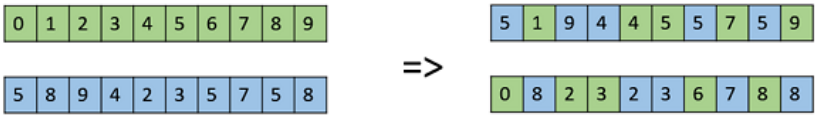
6.2 Multi Point Crossover

Multi point crossover is a generalization of one-point crossover wherein alternating segments are swapped to get new off-springs.



6.3 Uniform Crossover

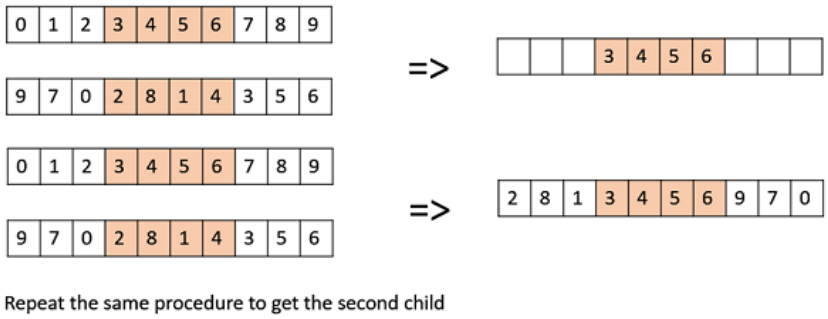
In a uniform crossover, we don't divide the chromosome into segments, rather we treat each gene separately. In this, we essentially flip a coin for each chromosome to decide whether or not it'll be included in the off-spring. We can also bias the coin to one parent (perhaps the one with the higher fitness), to have more genetic material in the child from that parent.



6.4 Davis' Order Crossover (OX1)

OX1 is used for permutation based crossovers with the intention of transmitting information about relative ordering to the off-springs. It works as follows

- Create two random crossover points in the parent and copy the segment between them from the first parent to the first offspring.
- Now, starting from the second crossover point in the second parent, copy the remaining unused numbers from the second parent to the first child, wrapping around the list.
- Repeat for the second child with the parent's role reversed.



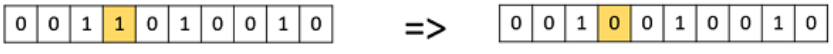
7 Mutation

In simple terms, mutation may be defined as a small random tweak in the chromosome to get a new solution. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability. If the probability is very high, the GA gets reduced to a random search. Mutation is the part of the GA which is related to the exploration of the search space. It has been observed that mutation is essential to the convergence of the GA while crossover is not. Again, the mutation probability, or mutation rate, relates to the famous exploration vs exploitation problem. A higher mutation rate leans towards exploration, and a lower towards exploitation.

Just like crossover, the mutation operator we use depends on the type of problem and chromosomes we have.

7.1 Bit Flip Mutation

In this bit flip mutation, we select one or more random bits and flip them. This is used for binary encoded GAs.

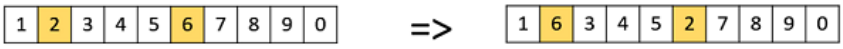


7.2 Random Resetting

Random resetting is an extension of the bit flip for the integer representation. In this, a random value from the set of permissible values is assigned to a randomly chosen gene.

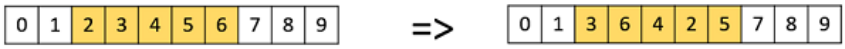
7.3 Swap Mutation

In swap mutation, we select two positions on the chromosome at random, and interchange the values. This is common in permutation based encodings.



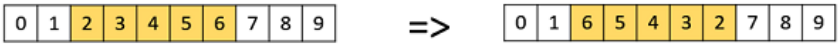
7.4 Scramble Mutation

Scramble mutation is also popular with permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.

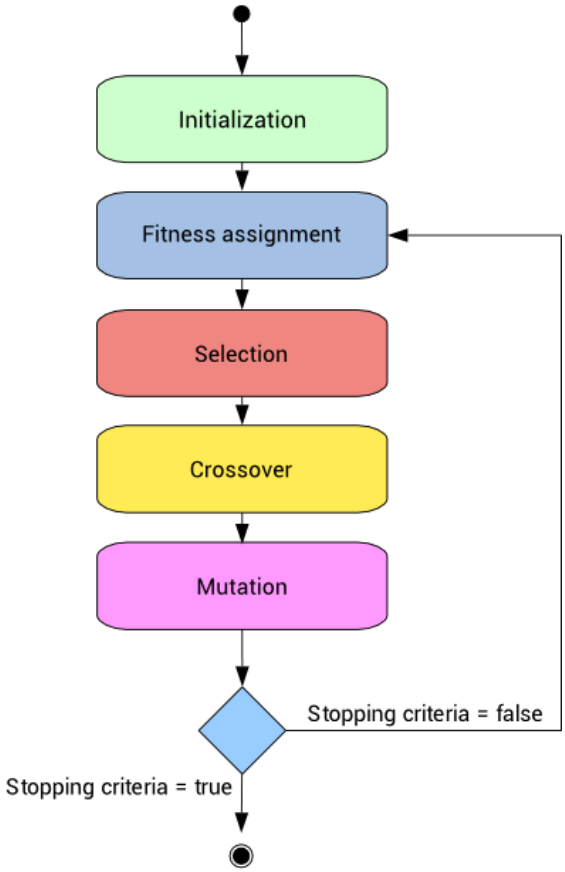


7.5 Inversion Mutation

In inversion mutation, we select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely invert the entire string in the subset.



8 Loop



We keep repeating this process until the stopping criteria is met. The stopping criteria may be when the maximum fitness of the population reaches a certain value, when a certain number of generations have passed, etc. It really depends on what goal we're trying to achieve with the GA.

9 Disadvantages

Sometimes, however, it's not appropriate to use a GA and other optimization searches may be better choices.

- need to tune hyperparameters (discard rate, mutation rate, crossover operator, etc.)
- not guaranteed a global maximum, may get stuck on local maxima
- time taken for convergence
- initial population may affect quality of solutions
- difficult to choose a good structure for some problems
- highly dependent on the definition of the fitness function

10 Application

GAs can be applied to

- Economics
- Neural Networks
- Image Processing
- Vehicle Routing
- Robotics
- Engineering Design
- and more.....

In summary, many scenarios calling for an indefinite output optimized to definite parameters. GAs can optimize these parameters to maximize the output.

GAs are very easy to implement and are used in a variety of fields, not just by data scientists. People often use DEAP (Distributed Evolutionary Algorithms in Python), which is an evolutionary computation framework for rapid prototyping and testing of ideas. It already incorporates the data structures and tool required to implement the most common evolutionary computation techniques: GAs, genetic programming, evolution strategies, particle swarm optimization, differential evolution, traffic flow, and the estimation of distribution algorithm.