Adaptive Genetic Algorithm Learning Agents: An Application in Finance (Lettau, 1997)

Tran Quang Thanh

January 23, 2022

The Jupyter Notebook code is available here on Github: Access

1 Theoretical Framework

1.1 Motivation

This paper investigates the decision-making of a bounded-rational agent in portfolio purchase. In this case, it is the number of risky assets one should hold. In theory, if the information is symmetric, a rational agent can easily solve the maximization problem and derive the optimal solution. However, in the real world, this is often impossible. Furthermore, since mutual fund investors are the least informed, the adaptive learning model can be more realistic than the rational model. An agent learns from observed outcomes of their investment decision and adjusts their portfolio composition accordingly.

One illustrious learning method is the Genetic Algorithm (GA), inspired by the evolution process observed in nature. GA is a way the genes (of a species) can transform themselves through many generations to survive and adapt to the new environment. In each generation, GA performs the following 3 operations.

First, GA defines fitness criteria to select the best-performing individuals. At the second step, GA performs a CROSSOVER operation. This is also called "mating" because the best individuals are paired randomly to mate with each other. After mating, offspring are born and inherit a mixture of genetic codes from their parents. For instance, imagine the genetic code of one individual can be represented by a binary string. If person 1 "101101" is to be mated with person 2 "011000", a crossover at point (3) slides each string in half and produces 2 new strings, "101000" and "011101". This process makes sure strong blocks of genetic code prevail in the long run. For the third operation, GA performs MUTATION. In our binary example, it is simply to flip 0 to 1 (and vice versa) with a given mutation rate. Mutation introduces new genetic materials, provides variety, and thus avoids the situation where evolution is stuck.

The GA framework is particularly interesting when being applied to the financial market. For the first operation, selection of the best fitted, one can think of criteria such as profit, or utility. An agent would want to hold the assets that generate a good payoff and discard the bad performing ones. For the second operation, the crossover is similar to trading. Two agents can trade the stocks from each other portfolios to see how they will perform. This action exploits rational belief. If 1 agent knows that the other has selected the best portfolio for him, the transaction is more likely to happen and acts as a diversification method. The mutation operation resembles an adventurous investment. The agent adjusts a small part of his portfolio component to see how it performs. This is also known as exploration. In reality, we have seen cases where conservative investors suddenly tried investing in new assets, such as Game Stop or Dogecoin. In a sense, it is not so different from mutation. Thus, adaptive learning with GA is a good simulation of the fund flows, as the paper proposed.

1.2 The Model

This section provides a standard model where an agent optimizes his assets holding to maximize a given objective function. The benchmark model assumes that an agent knows everything necessary so that the optimal solution can be calculated, whereas the GA model follows the solutions of adaptive learning agents.

We first examine the general framework for both models. There is one single asset whose value is normally distributed.

$$v \sim N(\bar{v}, \sigma^2)$$

where: v is the realized value, $\bar{v}$ is the mean or expected value of the asset and σ^2 as the variance.

The asset demand function for an agent is:

$$s = \alpha(\bar{v} - p)$$

where: s is the number of assets an agent wants, α is the demand parameter, p is the price of the asset. The price is assumed to be exogenous.

The net payoff w after the agent purchased s assets and its value v realized:

$$w = s(v - p)$$

From there, the agent derives a utility:

$$U(w) = -e^{-\gamma w}$$

where w is the net payoff, γ is a constant absolute risk aversion coefficient.

The problem is thus reduced to the choice of s such that U is maximized.

1.2.1 Benchmark Model

In a benchmark model, the optimal portfolio decision is a linear function of the mean value of the asset $\bar{v}$ and the price p:

$$s^* = \alpha^*(\bar{v} - p)$$

where:

$$\alpha^* = \frac{1}{\gamma\sigma^2}$$

After implementing the GA adaptive learning model, we will be able to see how close the adaptive learning behavior gets to this optimal solution.

1.2.2 GA Model

Although the choice of α in the benchmark model is straightforward, some maneuvers are needed to transform the problem of choosing α workable into a GA framework.

Assume that each agent has 1 portfolio profile. Each profile is represented by a binary string of length L. Let $\mu_i \in (0,1)$ be the i^{th} bit of the string, then such a string can be decoded into parameter α with the following formula.

$$\alpha = MIN + (MAX - MIN)\frac{\sum_{i=1}^{L} \mu_i 2^{i-1}}{2^{L-1}}$$

where $[MIN, MAX]$ is a arbitrarily chosen bound for α.

Crossover and Mutation operations will be performed on the bits of the string representing an α.

Assume that before the first crossover and mutation occur, that is, before an agent decides to trade with another or alter his first portfolio profile, he observed the performance of all available portfolios and then selected the best one for him. To choose a suitable portfolio, he needs a criterion of fitness selection, which can be defined as follows.

$$V_j = \sum_{i=1}^{S} U_j(w_j)$$

where V_j is the valuation of portfolio j, S is the number of times an agent observes how portfolio j performs, $U_j(w_j)$ is the utility from 1 realized payoff from 1 observation. Thus, one portfolio's value is the accumulative utility from S observations.

Assume that there are J agents in this market and the original number of portfolios is $2J$. Then after the first selection, we have the best J portfolios ready to be applied into the GA framework. Assume an agent's life is T. Then at each time t in T, the following steps happen.

1. Each agent observes the population of portfolios perform for S times. α is decoded into s. For each time (in S), the agent sees a different value of v (normally distributed), which implies a different value of w and U.

2. Each agent then selects the best portfolio for him based on criteria V (accumulative U after S observations)

3. The portfolios are randomly paired to perform crossover and mutation. Note that the original pairs are preserved. The new population is now updated with J parents and J offspring.

4. Repeat step 1 until T.

To settle the simulation, we employ some restrictions on GA. The crossover operation occurs with a constant probability CROSS, while the mutation rate decays over time with a half-life equal to 250 periods.

2 Code

The variance σ^2 is as high as the mean value $\bar{v}$, showing that the assets have high risk. The code follows closely the steps described in the last section.

2.1 Preparation

```
import numpy as np
import random
import matplotlib.pyplot as plt

v_bar = 1    # mean value of utility
var = 1      # variance of value function
price = 0    # price
T = 1500     # periods
S = 150      # observations
gamma = 1    # CRRA para
L = 20       # string length
J = 30       # population size
MIN = -4     # min value of alpha
MAX = 4      # max value of alpha
seq = [0,1] # possible bit in a string
iniPOP = 60 # initial population size
iniMUT = 0.08  # initial mutation rate
CROSS = 0.4 # crossover rate

# mutation rate decays exponentially with rate gMUT
halflife = 250
gMUT = np.log(0.5)/halflife
```

The parameters are set exactly to the paper. One difference is the number of periods. I set T to be 3 times larger so that we can see a convergence easier.

2.2 Step 1

```
""" A class of portfolios to store attributes"""
class Portfolio():
        def __init__(self, ruleStr):
                self.ruleStr = ruleStr
                self.alpha,self.s =self.find_values(self.ruleStr)
                self.V = 0 #default value for V
        # get the alpha and s of a portfolio
        def find_values(self, ruleStr):
                alpha = MIN
                for j in range(len(ruleStr)):
                        x = int(ruleStr[j])*(2**(j-1))/(2**(L-1))
                        alpha += (MAX-MIN)*x
                        s = alpha*(v_bar - price)
                return alpha, s
```

Here, since each portfolio profile is associated with its string (which is denoted 'ruleStr'), α, s, and V parameters, it is convenient to create a class of Portfolio to store the values. Since each agent retains one portfolio profile, 1 portfolio is analogous to 1 agent.

The function *find_values* takes in the string as input, decode and return 2 outputs, α and s. We do not want to invoke the valuation process after a portfolio is created, so let us set the initial valuation V of each portfolio to zero.

2.3 Step 2

```
def update_V(s):
        V = 0
        for i in range(S):
                d = random.gauss(v_bar, var)
                w = s*(d-price)
                u = -np.exp(-gamma*w)
                V += u
        return V
```

In Step 2, an agent needs to evaluate the portfolio, thus we need a function to do so. In the simulation, any time the evaluation is needed, this function will be called.

2.4 Step 3

```
# ------------ The GA functions ----------#
""" a function to select the best parents"""
def parent_select(population, n_parents):
        parents = []
        # sort the portfolios based on their V
        sortedPOP = sorted(population, key= lambda parent:
        ↪  parent.V, reverse=True)
        # select the first n parents
        for i in range(n_parents):
                parents.append(sortedPOP[i])
        return parents

""" a function to perform crossover between 2 parents """
def crossover(parent1, parent2, crossover_rate):
        length = len(parent1)
        # define the child strings
        child1 = ""; child2 = ""
        # perform under a probability CROSS
        if random.uniform(0,1) < crossover_rate:
                cross_point = random.randint(0, length-1)
                child1 = parent1[0:cross_point] +
                ↪  parent2[cross_point:length]
                child2 = parent2[0:cross_point] +
                ↪  parent1[cross_point:length]
        else:
                child1 = parent1; child2 = parent2
        return child1, child2
```

```
""" define a function to perform mutation on a string """
def mutate(child, mutation_rate):
    length = len(child)
    mutated_child = ""
    # perform under a probability MUT
    for bit in child:
        if random.uniform(0,1) < mutation_rate:
            if bit == "1":
                mutated_child += "0"
            else:
                mutated_child += "1"
        else:
            mutated_child += bit
    return mutated_child
```

With the above 3 functions, GA operations can be performed. One caveat is that the mutation is applied to all populations (both the parents and the children strings), not just the newly created children strings. Next, we define another function to update the population for GA.

```
""" a function to perform genetic algorithm """
def ga_update(population, n_parents, mutation_rate,
↪  crossover_rate):
    parents = parent_select(population, n_parents)
    portfolioList = parents #keep the parents in the new
    ↪  population
    parents_pairs = random.sample(parents, len(parents))
    # crossover
    for i in range(0, n_parents, 2):
        child1, child2 =
        ↪  crossover(parents_pairs[i].ruleStr,
        ↪  parents_pairs[i+1].ruleStr, CROSS)
        child1 = Portfolio(child1)
        child2 = Portfolio(child2)
        portfolioList.append(child1)
        portfolioList.append(child2)
    # mutate all population
    new_portfolioList = []
    for portfolio in portfolioList:
        new_ruleStr = mutate(portfolio.ruleStr,
        ↪  mutation_rate)
        new_portfolioList.append(Portfolio(new_ruleStr))
return new_portfolioList
```

2.5 The Simulaton

Before the first GA occurs, an initial population of portfolios must be created. From this pool of portfolios, 30 best-performing ones are selected and held by agents. After that, the first GA starts and the cycle repeats itself.

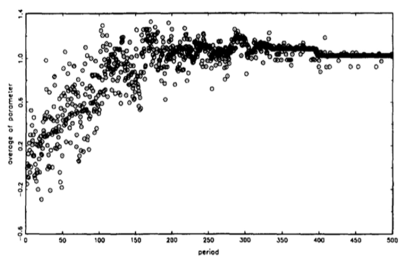
```
# ------------ The Initiation ----------#
# a function to create a random population of string
def gen_pop(string_length):
        ruleStr = ''"
        for i in range(string_length):
                ruleStr+=str(random.choice(seq))
return ruleStr

# initialize a random population of portfolios
original_portfolioList = []
for i in range(iniPOP):
        ruleStr = gen_pop(L)
original_portfolioList.append(Portfolio(ruleStr))

# before simulation, select the parents
for portfolio in original_portfolioList:
        portfolio.V = update_V(portfolio.s)
        portfolioList = parent_select(original_portfolioList,
        ↪  30)
```

Then, we run GA for T times and plot the results.

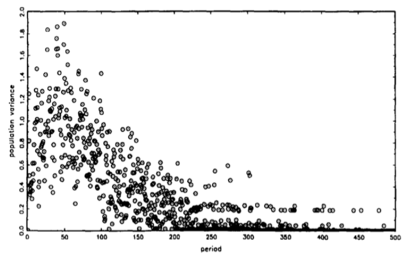
```
# plot elements
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
# runs the simulation for T times
for t in range(T):
        MUT = iniMUT*np.exp(gMUT*t)
# evaluate each portfolio
for portfolio in portfolioList:
        portfolio.V = update_V(portfolio.s)
# update population using GA
portfolioList = ga_update(portfolioList, J, MUT, CROSS)
# extract average parameters
alphaList = [portfolio.alpha for portfolio in portfolioList]
alphaMean = sum(alphaList)/len(alphaList)
alphaVar = np.var(alphaList)
# plot
ax1.scatter(t, alphaMean, color='blue', marker='.')
ax1.hlines(y=1, xmin=0, xmax=t, colors='red',
↪  linestyles='solid')
ax1.set_title('average alpha')
ax2.scatter(t, alphaVar, color='green', marker='.')
ax2.hlines(y=0, xmin=0, xmax=t, colors='red',
↪  linestyles='solid')
ax2.set_title('variance alpha')
plt.tight_layout()
```

3 Results

First of all, let us take a look at the simulation results from the paper.



(a) GA's average α



(b) GA's α variance

Figure 1: Simulation results from the paper, T=500

And our simulation:

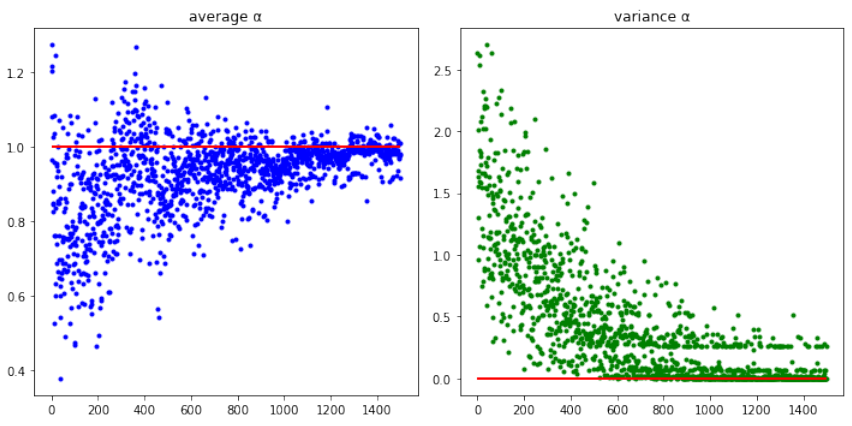


Figure 2: Author's replication 1, T=1500

Despite the nature of randomness, running again shows similar results.

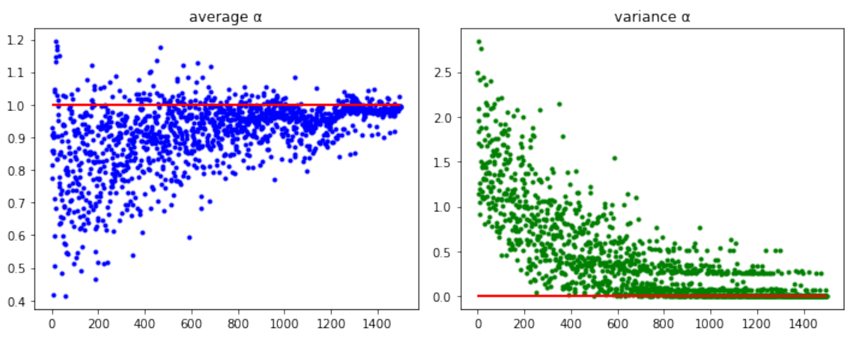


Figure 3: Author's replication 2, T=1500

The simulation shows that although the algorithm is approaching the optimal solution, the average number lingers a little above the optimal α^* in the benchmark model, which is indicated as the red line. The intuition is that when information is not enough, then agents tend to hold more assets than optimally needed. The author further argues that a piece of information that is crucial to the adaptive learning agent here is observations S. The bias to hold more risky assets tends to increase if the agent has to make a portfolio decision after observing its performance for only a short time. I confirm such a phenomenon by the following simulation.

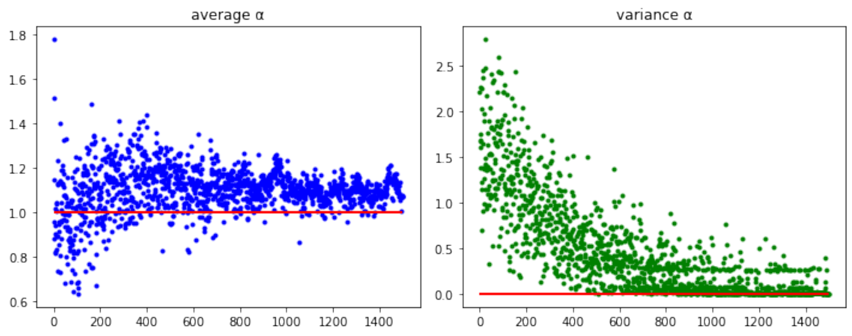


Figure 4: Author's replication 3. Parameter S changed from $150 \to 50$

By randomness, if an agent observes more positive than negative events, he tends to hold more risky assets. And since other agents are doing the same, this creates a bubble where risky assets are favored and trade exclusively. The bias is reduced if an agent is more patient to observe assets realization more times, before updating the portfolio decision.

The results are interesting for 2 reasons. First, it shows that an adaptive learning agent may never reach the optimal value if the observation of assets realization is not enough. This can explain partially the build-up of a bubble. Agents are not rational enough and information is not sufficient so they take rare events (high returns for risky assets) not so correctly, compared to the rational agents. A bubble might form if all agents behave in such a manner. Second, the framework of GA in the natural evolution process has been nicely introduced and applied in the social world, which in this case is the financial decision. An intuitive GA implementation here shows that a GA model is more accessible and realistic than many rational agent models. From this initial work, we can think of many expansions to simulate more complicated models, such as speculative bubbles, or propagation/spillover effects in a network.

4 References

Lettau, M. (1997). Explaining the facts with adaptive agents: The case of mutual fund flows. Journal of Economic Dynamics and Control, 21(7), 1117-1147.

LeBaron, B. (2000). Agent-based computational finance: Suggested readings and early research. Journal of Economic Dynamics and Control, 24(5-7), 679-702.