Active guidance system for a finless rocket using Neuroevolution
Faustino J, Gomez and Risto Miikkulainen

- Paul Neculoiu -

Introduction

Invented by the Chinese in the 12th century, rockets are probably mankind’s best invention since sliced bread. Applications are numerous but the most important are those aimed at space exploration and, to some extent, upper atmosphere scientific measurements like meteorology, environmental studies or astronomy. The latter have been in use since before the start of the space race and are still widely used today, as they provide one of the most cost-effective platforms for such experiments. These types of rockets are called sounding rockets.

Sounding rockets are meant to be cheap launch vehicle. This is mainly achieved by deploying cost-effective solutions in developing, building and operating the rocket such as by keeping the design as simple as possible. This often means discarding complex thrusters, sensors, performing detailed atmospheric studies and in-depth analysis of rocket dynamics, all of which would require resources that would greatly increase the rocket’s cost, thus kind of beating the purpose. Rather, design effective solutions such as fins for stabilization and simple feedback mechanisms are used. This leads to a robust, platform that can be reused for various missions without the need for refitting before each deployment.

Robust designs, however, have limited efficiency. Fins are easy to design and attach to a rocket, the physics are clear and the math is simple. However, the way in which they function is highly inefficient. Fins add weight to the rocket and create drag which effectively slows down the rocket and reduce its maximum attainable height or range. Designing a finless rocket is no easy task, as rocket flight dynamics are highly non-linear and require rigorous design methods requiring extensive engineering knowledge. The paper “Active guidance system for a Finless Rocket using Neuroevolution” by Gomez and Miikkulainen proposes a novel approach to the problem by use of genetically evolved artificial feedforward neural networks for the feedback-control loop. They claim that the use of their enforced subpopulations training method is able to accurately learn the control function. Their findings and approach will be presented in the upcoming sections.

The Problem

Rockets function by expelling accelerated mass, carried as fuel onboard the rocket, at one end of the rocket, thus propelling the rest of the rocket’s body as a result of the conservation of momentum. This leads to rockets acting, to some extent, like an inverse pendulum thus being inherently unstable.
This is corrected by use of more than 1 thruster, the output of which will be altered to change the rocket’s angle of attack through thrust vectoring.

The second stability issue stems from the atmospheric irregularities. When the rocket encounters such irregularities, the difference in pressure between zones translates into a force on the rocket making it change pitch or yaw and spin around the center of gravity. Combined with the fact that traversing the atmosphere at higher and higher speeds moves the pressure point on which most of the forces are exerted higher and higher on the rocket. If it is much higher than the rocket’s center of gravity, any minute irregularity may send the rocket on a crash course with disaster. Fins passively lower the center of pressure with regard to the length of the rocket, even lower than the center of gravity thus making the rocket stabilize itself with minimal, if any, active control.

As the rocket reaches higher altitudes, the atmosphere thins and drag decreases and both the fins and the atmosphere itself have less of an effect on the rocket. With less drag involved, the rocket is able to travel faster and higher requiring less control, but now, at an increased speed, even minute deviations from the course are potentially fatal for the mission.

While simplistic and effective, fins have one serious drawback. The extra drag exerted on the end of the rocket to stabilize it, is exerted on the entire rocket as a system, thus slowing it down and lowering its maximum potential. Eliminating the fins altogether and achieve a better efficiency is not a trivial task. There are a great number of variables that need to be taken into design consideration when dealing with finless rockets. The control function is highly nonlinear and the dynamics are very complex. The computations would require substantial effort to accurately solve the control problem.

The Solution

Gomez and Miikkulainen propose that the control issue can be resolved by simply mapping the sensor readings to the thruster output through the use of an artificial feedforward neural network, eliminating the need for analytical knowledge of the respective rocket dynamics or the appropriate control strategy to deploy. The claim is that such a network can be evolved using a genetic algorithm they dubbed “Enforced Subpopulations”. With sufficient training sessions the network can learn the sensor-output mapping provided a sufficiently accurate environment (preferably a sufficiently accurate simulator to reduce costs) is present in the training.

Feedforward neural networks

A feedforward neural network is an artificial neural network that performs a nonlinear function of the weighted sum of the inputs neurons within the neurons of an intermediate layer of neurons (the hidden layer) before being mapped out to the output units (or the next hidden layer if there are several) as a weighted sum of the hidden units. Typically the nonlinear function used is of a sigmoidal form (sigmoid, arctangent, hyperbolic tangent, etc). The purpose is to mimic a step function in a continuous
manner as step functions have no derivative in their transition phase and are, thus, unsuited for continuous tasks. Sigmoidal functions are also near-linear close to the origin and, with small enough weights on the input units, can also mimic linear functions rather accurately.

\[ o_k = g \left( \sum_{j=1}^{m} f \left( \sum_{i=1}^{n} x_i w_{ij} \right) q_{jk} \right) \]  

(1)

Theoretically, given enough hidden units, feed forward networks can approximate virtually any nonlinear function. The key issue is setting the right weights. There are several methods by which accurate weights can be determined and the subject has been researched extensively over the years. One of the most versatile methods (albeit rather slow) is by means of genetic programming.

**Genetic methods**

The principles of genetic programming state that any function can be optimized by applying successive genetic operators to members of a population of sub-optimal solutions. The function’s parameters that are trying to be optimized are encoded in a sequence of genes, forming a chromosome that defines a solution. Multiple possible solutions exist at one time and it is the job of the genetic operators to perform a rather pseudo-random search of solution space. Typical operators are those derived from biology that push forward the evolution of species:

- **Crossover**: two solutions swap a certain number of genes belonging to the function the population is trying to optimize in a discrete manner (parameters describing the same feature of the function are linked together. It is usually not practical to put a foot gene in the gene space that describes hands as the parameters describe different features and are incompatible) and thus a new possible solution is born. Sometimes it might prove useful to encode the different genes of different features of the solution in different chromosomes and exchange whole chromosomes between members instead of individual genes (exchange the whole hand data, not just individual fingers).

- **Mutation**: each new solution has a chance to slightly change one of the genes, either flipping one or more bits if the features are binary or altering the gene by adding a certain amount of noise (typically Gaussian-shaped distributed).

- **Selection**: solutions are evaluated for their potency of solving the problem at hand and the good solutions are kept for the next batch of optimization while bad solutions are discarded and replaced with new ones. Populations are typically maintained constant.

This is by far not the only way of applying genetic algorithms. Genetic methods can be vast and inherently complex. However, these are the basic three methods which are almost always applied, either as described above or as a variation of them.

Genetic (or evolutionary) algorithms applied to neural networks form the sub branch of neuroevolution. Neuroevolution attempts to optimize a neural network for its classification or regression task by evolving the network’s parameters, the weights. Typically, the network’s weights,
both those incoming from the input layer to the hidden layer, as well as those going from the hidden layer to the output layer, are encoded in a gene sequence either as a single chromosome for the whole solution, or as different chromosomes, one for each hidden neuron and preserve the function each hidden neuron performs over generations.

The advantage of neuroevolution over other methods of training neural networks lies in the fact that no knowledge of the solution space is needed and can be directly applied to any neural network problem. The downside, is the long computation time required to perform the training, as numerous possibilities and solutions need testing repeatedly and redundancy in solutions is bound to arise. In fact, if genetic diversity is not sufficiently preserved, by more or less artificial means, then the entire population risks getting stuck in a local optimum and evolution ceases. Large populations and computational artifacts, such as population resets, spontaneous mutations, soft selection, etc. are required in order to keep the diversity high. Still, 4 billion years of evolution have proven that the method is computationally sound.

**Enforced Subpopulations (ESP)**

Typical neuroevolution algorithms encode the whole network as an entity within a population of possible neural networks and attempt to evolve the whole thing. The downside is that training may take a long time and networks that arise may be specialized to certain features, depending on the current state of the testing environment.

In their paper, Gomez and Miikkulainen suggest that each hidden neuron be treated as a separate evolutionary entity and evolve them separately while testing the performance of the whole network. Instead of a number of possible neural network solutions being evolved, they attempt to evolve each hidden neuron (somewhat) independently. Each of the weights going into a hidden neuron from the inputs as well as the weights going out of the neuron towards the outputs are encoded into a gene and a population of such entities is generated, tested and evolved. The process is repeated for each of the network's hidden neurons and these subsets are put together to form a complete network. This should lead to neurons evolving independently and specialize rapidly into good network subfunctions. EPS evolution proceeds as follow:

1. **Initialization.** The number of hidden units $h$ is specified and a subpopulation of chromosomes is created for each unit. Each chromosome encodes the input and output weights of the unit with a random array of real numbers.

2. **Evaluation.** A set of $h$ neurons is selected randomly from each subpopulation and form a network hidden layer. The method is tested and evaluated (in this case by flying the rocket and seeing how high it got) and each neuron receives a fitness score. The respective score is added to a cumulative fitness of each neuron that participated. The process repeats until each neuron has participated in an average of e.g. 10 trials.
3. Recombination. The average fitness of each neuron is calculated by norming the cumulative fitness to the number of trials in which it participated. Neurons are then ranked by fitness within each subpopulation and each neuron in the top quartile is recombined with a higher-ranking neuron using 1-point crossover (the chromosomes are split into 2 parts and those 2 parts are swapped between parents to produce offspring) and mutated at low levels to create a new solution to replace the lower half of the subpopulation.

4. The cycle is repeated until a network that performs sufficiently well in the task is found. (in this case, the rocket reaching burnout as high as possible).

Evolving networks at a neuron level eliminates the need for species to organize themselves out of a single large population, and the progressive specialization is not hindered by recombination across specializations that fulfill somewhat orthogonal roles (if in one network, neuron 1 is dominantly responsible for controlling thruster one, while in another network it evolved as more responsible for controlling thruster 2, a potential offspring can control either thruster 1, 2, both or neither. By enforcing subpopulations, one eliminates invalid combinations, thruster 1 solutions always combine with thruster 1 solutions and eliminates most of the less relevant solutions from appearing at all).

However, this fast specialization also causes diversity decline over the course of evolution, which may leave possible solution spaces unexplored. After a population has converged to a solution it can no longer readapt to a new task. The solution found to deal with premature convergence is combining ESP with burst mutation. Once performance has stagnated and the population is no longer evolving, everything but the best fitting solution in the population is killed off, and replaced with mutations of the best solutions by adding Cauchy distributed noise:

\[ f(x) = \frac{\alpha}{\pi(\alpha^2 + x^2)} \] (2)

This recharges the population and keeps diversity in check. As opposed to a Gaussian distribution, the Cauchy distribution has a wider spread of values, such as, while most of the new solutions are still close to the known maximum, quite a few will spawn up pretty far away.

The experiment

Sadly, the experiment was not performed on an actual rocket (RSX-2 sounding rocket), but on an open source rocket simulator (JBSim rocket simulation). The rocket was first flown with full fins and no guidance and half fins and no guidance as a benchmark. Afterwards, the fins were reduced to ¼ of their normal size and rocket control based on the neural network was added. This was done in order to perform incremental evolution, such that the initial task of flying directly finless was too complicated for the network to resolve.

The rocket controller is represented by a feedforward neural network with one hidden layer. Every 0.05 seconds the controller reads the sensor inputs which provide information about the current state of the rocket (pitch, yaw, altitude, throttle power, etc). The input vector is propagated through the
sigmoidal hidden and output units of the network and thus produce a new throttle position for each engine. The maximum throttle change from maximum throttle is limited to 10%. This value was determined to be sufficient during early testing. The controller is initiated with 10 hidden units representing subpopulations of 200 neurons. These then wire up randomly with other neurons from different subpopulations to produce a controller solution. The simulation flight is then performed and the altitude attained is evaluated. If at any point, the angle of attack of the rocket or the sideslip angle exceed +/- 5 degrees, the rocket is considered to have tumbled out of control and is thus a catastrophic failure. Once sufficient evaluations have been performed, the populations are evolved to the next generation.

Results

After some 600,000 such evaluations, the ESP evolved controller succeeds in flying the ¼ finned rocket to burnout. Some 50,000 evaluations later the transition to a finless rocket is also made possible. Without guidance the full finned rocket reaches burnout at about 70,000 ft, while rockets with smaller fins and no guidance fail before reaching burnout. Using the neural network guidance, the ¼ finned and finless rockets reach burnout and exceed the full finned rocket's altitude by 10,000 and 15,000 ft, which is some 20% increase in altitude, which is quite an achievement considering rocket costs.

Discussion

The rocket problem is just a representative example of a non-linear problem that neural networks can resolve and shows that the ESP algorithm is capable of training such a network for a high-dimensional non linear control task without prior engineering knowledge of the system.

There is one minor flaw, though, in the algorithm's design. The crossover method between neurons is 1-point crossover. This splits the chromosome in 2 parts which are then mixed. However, this leads to a certain spatial correlation between input units. If the pitch is next to the yaw on the input vector, then they shall have genes close to each other and will most likely be cut and pasted together. This might not be desirable as pitch and yaw might not influence the same characteristics and evolution will, in fact, get slowed down as variations between pitch and yaw have to wait for mutations to occur, mutations which might not lead to a directly better performing controller and thus die out.

Even though the system was not tested on an actual rocket (which is regrettable), it does lie within the authors’ future plans, applications of the ESP algorithm may span to other non-linear domains.