# Control Freak

## Solving Classic Control tasks using ES and NN

Tzoof Avny Brosh

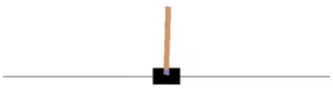
# 1. Classic Control Tasks

Control theory problems from the classic Reinforcement Learning literature.
The problems are simulated using a framework built by OpenAI called Gym.
For each task, the environment provides an initial state from a distribution (so each game might be a little different), accepts actions, and given an action provides the next state and a reward.
Each states and actions simulates real physics behavior.
Example of states values are location, velocity, angle of joint, angular velocity and momentum, actions are usually the force that should be acted on an object.
The goal is to build a policy model that given a state returns the action that will lead us to the highest reward.

# 2. The tasks

## Cartpole

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart.
The pendulum starts upright, and the goal is to prevent it from falling over.
A reward of +1 is provided for every timestep that the pole remains upright.
The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

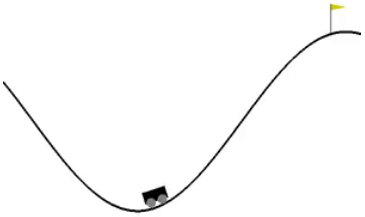


## Acrobot

Acrobot is a 2-link pendulum with only the second joint actuated. Initially, both links point downwards. The goal is to swing the end-effector at a height at least the length of one link above the base. Both links can swing freely and can pass by each other, i.e., they don't collide when they have the same angle.

### MountainCar

A car is on a one-dimensional track, positioned between two "mountains". The goal is to drive up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum.



# 3. Related work

A standard reinforcement learning setting is formalized as a Markov Decision Process (MDP) and is made of an agent interacting with an environment E over a disceate number of time frames. At each time frame t, the agent receives a state s(t) and reacting action a(t) taking the agent into state s(t+1) and rewarding the action with reward r(t+1) The process continues until the agent reaches a terminal state or the time runs out, marking the end of an episode. The goal of the agent is to maximize the expected total reward.
(Lillicrap, T.P et. Al 2015) suggest implementation of RL using deep learning network to solve continuous control problems.
(Salimans et al. 2017) suggest using evolution strategies when using reinforcement learning (RL) thecniques as it is easy to implement , reducing the need for back propagation, and enabling parallelization and increasing the robustness of the solution.
(Khadka and Tumer, 2018) combines Evolutionary Strategies algorithm and a Deep RL agent by first training some RL actors and then periodically injecting gradient information into the EA population.

(Müller, N. and Glasmachers, T., 2018) in their work investigate the utility of different algorithmic mechanisms of evolution strategies for problems with a specific combination of challenges, namely high-dimensional search spaces and fitness noise with evaluation for three different algorithms and problems (acrobot, bipedal walker, and robopong). Their

main findings are that adaptation of the mutation distribution is less valuable in high dimensions because it kicks in only rather late.
(Colas, C., Sigaud, O. and Oudeyer, P.Y., 2018) define a new framework called GEP-PG for "Goal Exploration Process - Policy Gradient". First the use Goal Exploration Processes (GEPs) to efficiently explore the continuous state-action space, then the resulting samples are stored into DDPG (the deep RL algorithm), which processes them to perform sample efficient policy improvement.
They evaluated the proposed solution using two benchmarks showing distinct properties: the Continuous Mountain Car and Half-Cheetah (HC).
In our work we used Openai Gym, a toolkit for developing and comparing reinforcement learning algorithms and DEAP library as an ES infrastructure.

# Methods and algorithms employed.

The algorithm:

The genotype: a list of floats.
The phenotype: the weights of a policy neural network model.

Evolution strategy with CMA

Using fitness maximization method

- Population size = 1720
- Initial sigma = 5
- Num of generations = 300
- Initial centroid = 5

Neural network

- Consist of 2 layers.
- Layer 1 –
    - Input size is set automatically according to the state size defined in the environment properties.
    - Output size 12
    - activation: tanh
- Layer 2
    - Input size 12

    - Output size is set automatically according to the number of actions defined in the environment properties.
    - activation: tanh
- Total number of weights - 75- 123, depend on the environment.

# 4. Software overview

## TrainES class

We used the Deap python library.
A generic class for training evolution strategy algorithms.
The class trains a list type individual.
Receives a fitness class that defines the size of the individual and calculates the fitness for an individual.
The class also handles experiments managements and saves logs for each experiment.

Properties: params (hyper parameters) and the Fitness object
Functions: train function

## Fitness abstract class

An abstract class for fitness evaluation, defines everything you need to implement in order to use the TrainES class.
Defines the size of the individual (num_features) and the game_name (for logging)
Calculates the fitness for an individual through the evaluate_task method.

## GameFitness class

A generic class for fitness evaluation in gym environment.
Calculates the fitness for an individual.
The fitness is calculated by creating a NN, initiating its weights from the individual values, and then running an episode of the game using the NN.
The running process:
4.1. It receives a random state from the environment
4.2. Selects an action by feeding it to the NN
4.3. Receives a new state and a reward from the environment
4.4. Repeats a-c until the end of the episode.
The fitness is the sum of the rewards in an episode (or the average sum of rewards if running several episodes).

Properties:
Game properties: game name, gym environment object, num of episodes to run per fitness.

NN size properties: initiated from the environment, according to the environment game specific configuration.
Policy object: a NN class initialized with the NN properties.

Functions:
evaluate_task - calculates the fitness for an individual.

### PolicyNetwork class

A neural network class for deciding on the next action given a state.

Properties:
NN size properties

Functions:
run_nn_get_action – choosing the next action given a state.
get_policy_weights_from_individual – initializing the NN weights given an individual

# 5. Work process

General challenge

Runtime: each training took a too much time (around 5 hours)

Solutions:

1. Reducing the size of the NN layers.
   This caused a decrease of the size of each individual – reducing the ES time.
   And also decrease in the weights of the network – reducing the network's computation time (the network is run for calculating the reward of each step in the episode – around 200 steps in each fitness calculation).

2. Implementing the NN ourselves.
   We first implemented the NN using Keras and later using TensorFlow.
   Both implementations took a lot of time to run.
   This is because the common NN frameworks optimize their performance to reduce the backpropagation time.
   Since we only need the forward process of the NN, implementing it ourselves reduces the overhead of the backpropagation and reduced the runtime.

# 5.1. CartPole

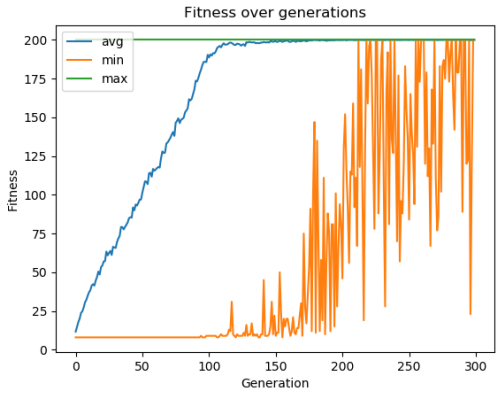
## First version

### Solution description

We first train with these initial hyperparameters.
Population size = 1720

- Initial sigma = 5
- Num of generations = 300
- Initial centroid = 5

### Population convergence graph




Final max score: 200
Final average score: 200
Training time: 30m

### Testing the best individual

We tested our best individual for 100 episodes of the task.

Avg score over 100 episodes: 45.84

Min score: 8.0, max score: 200.0

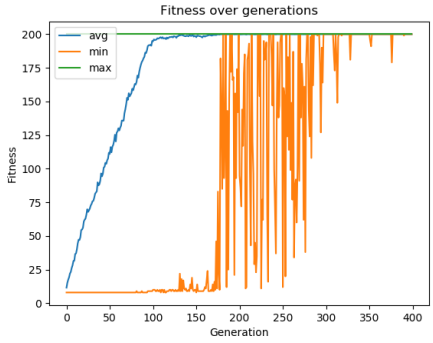
### The challenges we encountered

Since the initial state changes in each run, a good fitness in train night infer to a good score on a specific game and not on different games.

## Second version

We trained the model for more generations - 400 to increase the minimum population fitness:

### Population convergence graph:




Training time: 50m

### Testing the best individual

Avg score over 100 episodes: 63.89

Min score: 10.0, max score: 200.0

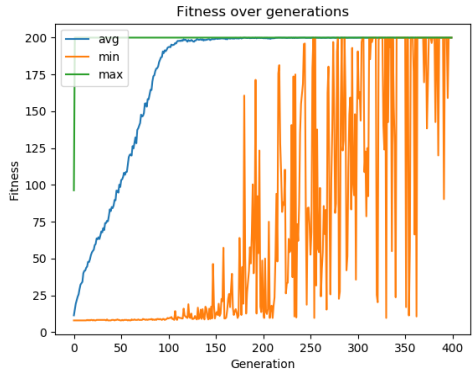
Better than before but still not good enough!

## Third version

We switched the fitness from score over 1 episode to average score over 3 episodes of the task.

It took more time to train – 5 hours instead of 30 minutes

### Population convergence graph:




### Testing the best individual:

Avg score over 100 episodes: 200.0

Min score: 200.0, max score: 200.0

# 5.2. Acrobot

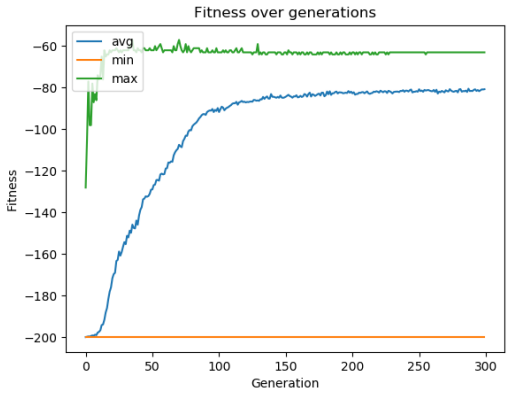
## First version

### Solution description

We first train with these initial hyperparameters.

- Population size = 2460
- Initial sigma = 5
- Num of generations = 300
- Initial centroid = 5

### Population convergence graph




Final max score: -62
Final average score: -83
Training time: **4h!**

### Testing the best individual

Avg score over 100 episodes: -85.46

Min score: -212.0, max score: -69.0

### The challenges we encountered

1. very high runtime-

   Since the maximum number of steps is 500 (in the other tasks where it is 200). The state size is 6, making the number of weights 108 (compared to 86 in cartpole and 75 in mountain car)

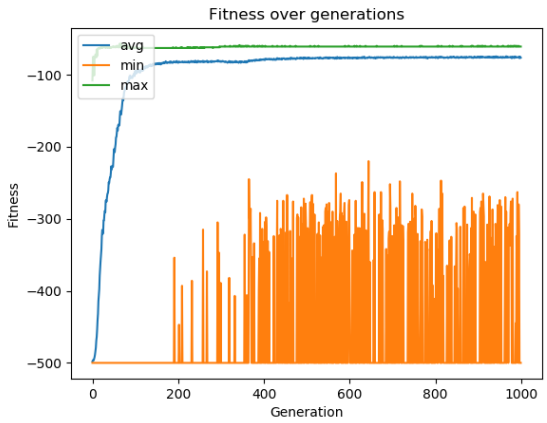
2. Differences between train and test time

Since the initial state changes in each run, a good fitness in train night infer to a good score on a specific game and not on different games.

## Second version

We trained the model for more generations - 1000 to increase the minimum population fitness:

### Population convergence graph




Training time: **10h!**

### Testing the best individual

Avg score over 100 episodes**: -80.3**

Min score: -135.0, max score: -64.0

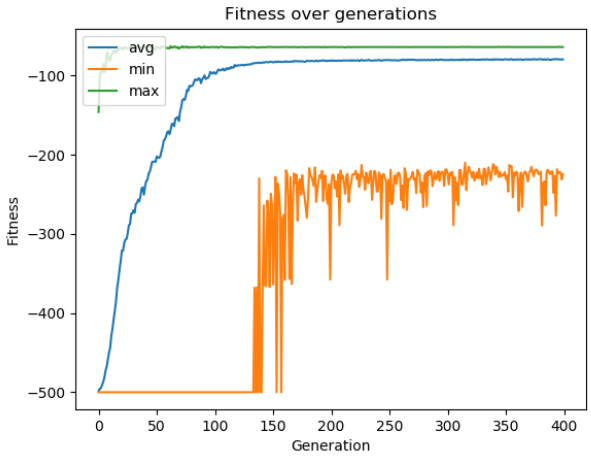
Better than before, and with lower variance.

## Third version

We switched the fitness from score over 1 episode to average score over 3 episodes of the task.

It took more time to train 1 generation.
And it took more generations to converge (which actually confirmed our theory that the score over one episode wasn't a sufficient fitness score)

### Population convergence graph:




Training time: **15h!**

### Testing the best individual:

Avg score over 100 episodes: -86.67

Min score: -187.0, max score: -61.0

We can see that the maximum score is higher but surprisingly the minimum score is lower and the average is also lower.

# 5.3. MountainCar

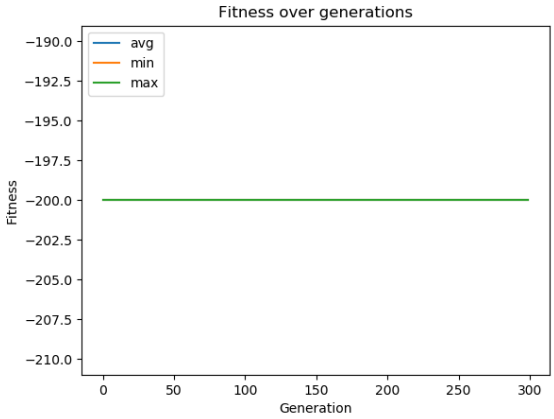
## First version

### Solution description

We first train with these initial hyperparameters.

- Population size = 1500
- Initial sigma = 5
- Num of generations = 300
- Initial centroid = 5

### Population convergence graph



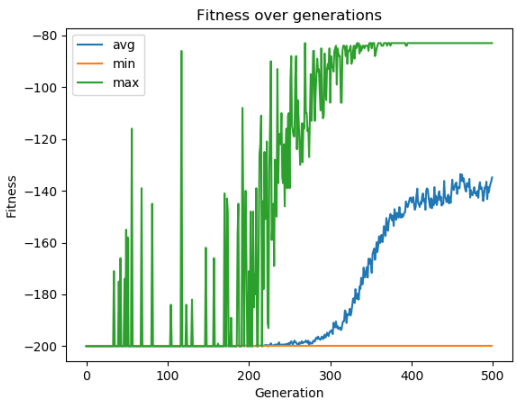
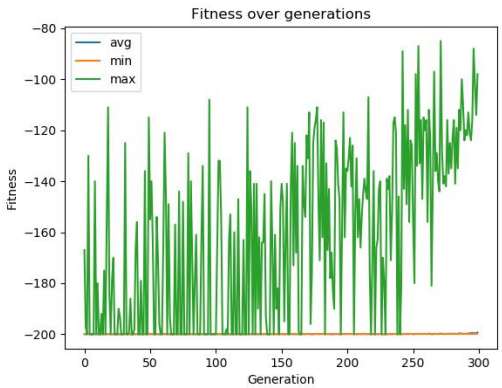

### The challenges we encountered

As you can see in the above graph. the main challenge in solving MountainCar is that no individual of our first generation reached the top of the mountain!
That means they all got the same fitness and had no way to improve, this continued for all generations that followed as you can see in this sad graph:

## Second version

We changed the initial centroid to 0 and increased the initial sigma to 50 and 20 to reach a higher variance.

### Population convergence graph

| Sigma =20<br>Number of generations = 500 | Sigma = 50<br>Number of generations = 300 |
|---|---|
|  |  |
| Train time: 30m | Train time: 86m |

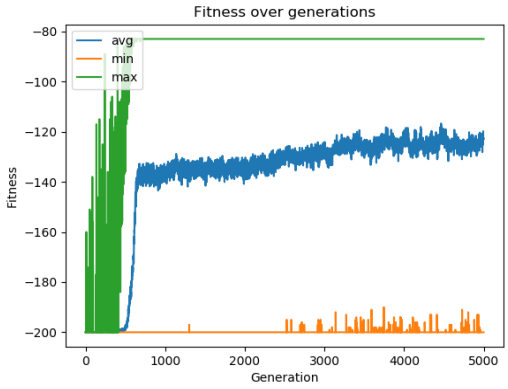
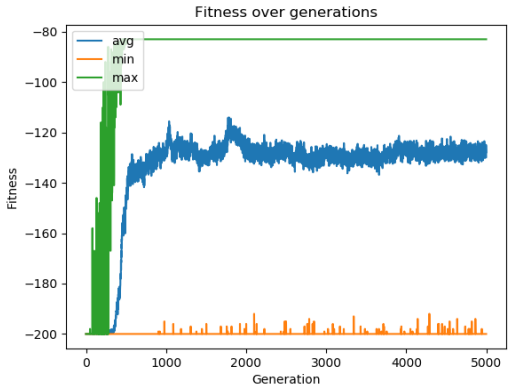
| | |
|---|---|
| Testing:<br>Avg score over 100 episodes: -138.35<br>Min score: -200.0<br>Max score: -83.0<br><br>Although its graph looks better, it received worse results at test time | Testing:<br>Avg score over 100 episodes: -127.78<br>Min score: -166.0<br>Max score: -84.0 |

## Third version

Increasing the number of generations to 5000, checking several centroid and sigma combinations, the best combinations were of initial sigma values of 20 and 50, and centroid =0.

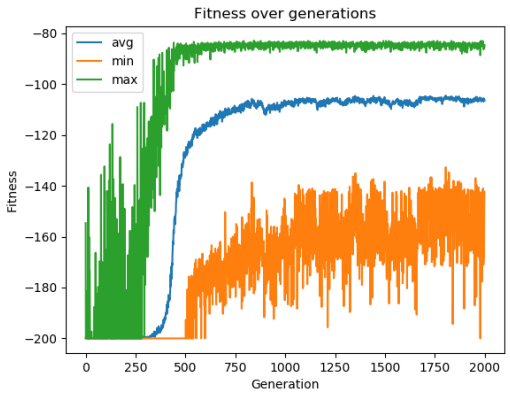
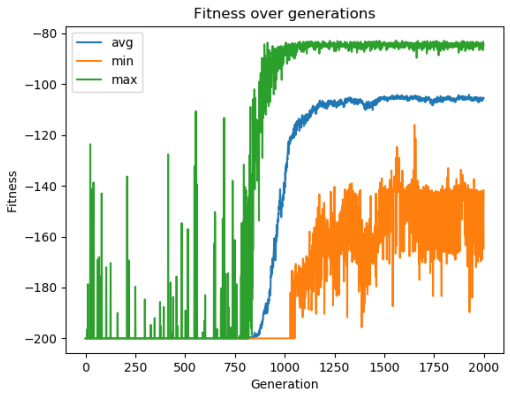
### Population convergence graph

Initial sigma value of 20 and 50 respectively

| Sigma =20<br>Number of generations = 5000 | Sigma = 50<br>Number of generations = 5000 |
|---|---|
|  |  |
| Train time: 3h | Train time: 3h |
| Testing:<br>Avg score over 100 episodes: -115.13<br>Min score: -173.0<br>Max score: -83.0 | Testing:<br>Avg score over 100 episodes: -125.48<br>Min score: -172.0<br>Max score: -83.0 |

## Forth version

In order to increase the performance at test time we wanted to increase the durability of each individual to different starting states.
We switched the fitness from score over 1 episode to average score over 3 episodes of the task.

| Sigma =20<br>Number of generations = 2000<br>Episodes per fitness = 3 | Sigma = 50<br>Number of generations = 2000<br>Episodes per fitness = 3 |
|---|---|
|  |  |
| Train time: 3h | Train time: 3h |
| Testing:<br>Avg score over 100 episodes: -111.41<br>Min score: -200.0<br>Max score: -83.0 | Testing:<br>**Avg score over 100 episodes: -106.26**<br>**Min score: -115.0**<br>**Max score: -83.0** |

# 6. Final results and conclusions.

## Final results

OpenAI published a score for solving each task here

Cartpole-v0

Solving - getting average reward of 195.0 over 100 consecutive trials.
Our Score - average reward of 200 over 100 consecutive trials.
**Solved!**

Acrobot-v1

Solving - Acrobot-v1 is an unsolved environment, which means it does not have a specified reward threshold at which it's considered solved.
But there is a leaderboard here

Our Score - average reward of -80.73 over 100 consecutive trials.
This brings us, as of 10.01.2020 to the **10th place in the leaderboard.**

MountainCar-v0

Solving - *getting average reward of -110.0 over 100 consecutive trials.*
Our Score - average reward of -106.26 over 100 consecutive trials.
**Solved!**

## Conclusions

We learned that a high average score of the population does not correlate with a high durability of its individuals to different instances of the game (different starting states). This caused us to change our fitness measurement score to an average of different episodes (games).
Also, we learned that for some tasks, like mountain-car it's very hard to even reach one individual that gets anything but the worst fitness available.

# 7. Bibliographic references.


1. Gym.ai wiki - https://github.com/openai/gym
2. Gym site - http://gym.openai.com/
3. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.
4. Khadka, S. and Tumer, K., 2018. Evolutionary reinforcement learning. arXiv preprint arXiv:1805.07917.
5. Müller, N. and Glasmachers, T., 2018, September. Challenges in high-dimensional reinforcement learning with evolution strategies. In International Conference on Parallel Problem Solving from Nature (pp. 411-423). Springer, Cham.
6. Colas, C., Sigaud, O. and Oudeyer, P.Y., 2018. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. arXiv preprint arXiv:1802.05054.
7. Salimans, Tim, et al. "Evolution strategies as a scalable alternative to reinforcement learning." arXiv preprint arXiv:1703.03864 (2017).
8. DEAP library https://deap.readthedocs.io/en/master/