InstaDeep™

Combining Neuroevolution and Feature Engineering for Improved Performance on the Frostbite Atari Game

Rihab Gorsane, Amine Kerkeni, Karim Beguir

Abstract

This work is an extension of recent work done by Uber AI Labs on Deep Neuroevolution demonstrating how gradient-free methods, such as population-based Genetic Algorithm, can learn to play Atari games from raw pixels [1]. Our work extends the baseline implementation presented by Uber AI Labs by adding further features to improve the performance. We validate the effectiveness of our chosen approach on the **Frostbite Atari game.** The results show that **our implementation outperforms the original algorithm.**

Background

Learning to act directly from raw pixels was challenging until RL algorithms harnessed the representational power of DNNs. Here, we integrate a GA and a DNN containing over 4M parameters [2], the largest network ever evolved with GAs. It has long been assumed that a simple GAs would fail at such a large scale.

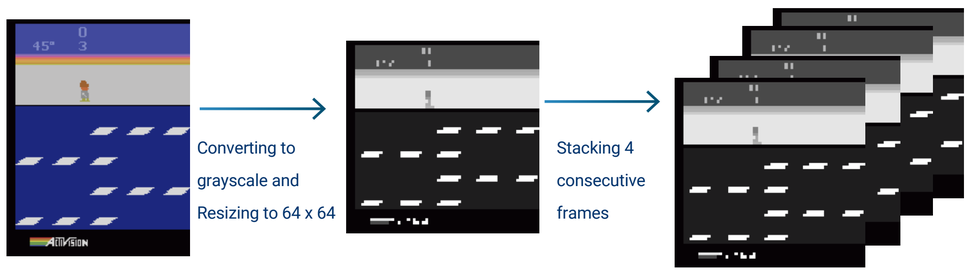
Algorithm

Algorithm 1 Genetic algorithm

Input: mutation function ψ, population size N, number of selected individuals T, policy initialization routine ϕ
for g = 1, 2, ..., G generations **do**
  for i = 1, ..., N in next generation's population **do**
    if g=1 **then**
      $P_i^g = \phi(N(0,1))$ ▷ initialize random DNN
      $F_i^g = F(P_i^g)$ ▷ assess its fitness
    else
      if i=1 **then**
        $P_i^g = P_i^{g-1}$
        $F_i^g = F_i^{g-1}$ ▷ copy the elite
      else
        k = uniformRandom(1, T) ▷ select parent
        $P_i^g = \psi(P_k^{g-1})$ ▷ mutate parent
        $F_i^g = F(P_i^g)$ ▷ assess its fitness
Sort P^g and F^g with descending order by F^g
Result: highest performing policy P_1^g

Environment

We apply the frame-skipping technique to accelerate playing (k=4)[3]. Then over two consecutive frames, we perform a pixel-wise maximum operation to handle flickering and detect movements of bright objects. Finally, we apply some preprocessing steps in order to create our state.

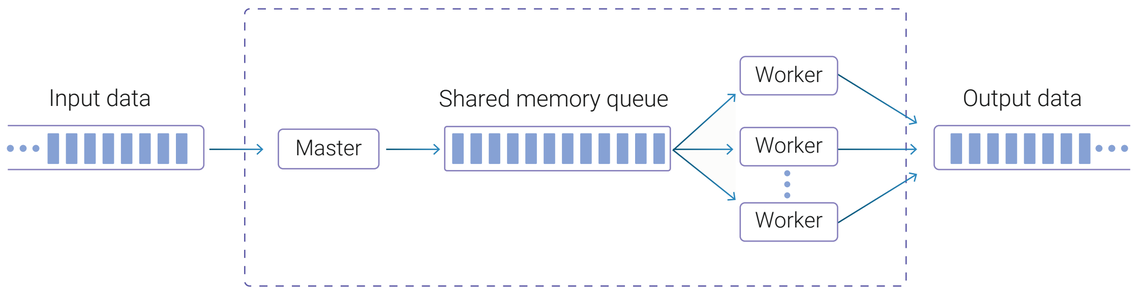


Model architecture

The final representation of the preprocessed state is fed into a Convolutional Neural Network (CNN) model as input. Given the **state S_t** as input and the parameters values of the model, the network is able to output the **action A_t** to take in that step t.

Distributed computation

The genetic run does less math than gradient-based methods but on many networks (order of 1k-5k networks simultaneously).This is computationally expensive. Therefore, we need to work with multiple machines and use the concept of a **task queue with master and workers on different machines.**

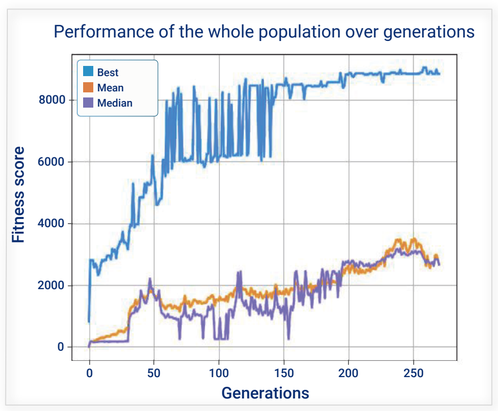


Model Compression

Traditionally, GA implementations store each individual as parameter vector θ. Since we have workers on several machines, we need to pass neural networks to each other. The traditional approach scales poorly in memory and network transmission costs with large population and parameter vector. The idea behind **compressing the model** is to store the parameter vectors by encoding each parameter vector θ to an initialization seed τ0 (which is used to initialize the network) plus the list of all random seeds τ1...τg that produced the series of mutations applied to θ in every generation.

Experimental Results

The agents are evaluated on a random number of no-op initial actions ranging from 0 to 30) to achieve some randomness in the environment. We note also that the number of game frames the agent has experienced over the course of a GA run is one billion frame.
During training, each agent is evaluated on a full episode capped at 20k game frames and the agent's fitness score is the final episode accumulated reward.



We record the best agent found in each of **3** independent, randomly initialized GA runs. The final score for each run is calculated by taking the highest-scoring elite across generations, and averaging the score it achieves on **200** independent evaluations. The final score for the domain is the median of final run scores.

	DQN	ES	RS	A3C	GA (Uber)	GA (Uber)	GA (InstaDeep)
Frames	200M	1B	1B	1.28B	1B	4B	1B
Time	~ 7-10d	~ 1 h	~ 1 h	4d	~ 1 h	~ 4 h	~ 36 h
Forward passes	✓	✓	✓	✓	✓	✓	✓
Backward Passess	✓	–	–	✓	–	–	–
CPU Cores					720	720	68
Frostbite	797	370	1379	191	4801	5623	**8270**

Results of DQN, ES, RS, A3C AND GA on Frostbite

Conclusion

With the architecture described above, our approach outperforms Uber's implementation, **doubling the fitness score on the Frostbite Atari game** while using the **same configuration**, **less computational power** and **also less training frames**. This encourages us pursue further research in this direction, as well as other game/test environments. We will also consider the use of the Anisotropic Self-Adaptive Gaussian mutation where a standard deviation is attached to each variable encapsulated in the parameter vector θ.

References

[1] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning, arXiv:1712.06567 [cs.NE] New York: Uber AI Labs, Dec. 2017
[2] Mnih V., Kavukcuoglu K., Silver D., Rusu A. A., Veness J.,Bellemare M. G., Graves A., Riedmiller M., Fidjeland A. K., Ostrovski G., and others. Human-level control through deep reinforcement learning. Nature, 2015.
[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, Playing Atari with Deep Reinforcement Learning, arXiv:1312.5602 [cs.LG]. Deepmind, Dec. 2013