Training agents in reinforcement learning with genetic algorithms

Semir Salkić
Faculty of Computer Science
University of Ljubljana
Ljubljana, Slovenia
Email: ss9343@student.uni-lj.si

***Abstract*—Reinforcement learning is an area of machine learning that involves training intelligent agents to take decision based on the reward function. Each agent is used to create optimal exploration vs. exploitation trade-off in an environment unknown to the agent. In Deep Q Learning networks (DQN) gradient - based learning algorithm is used for optimization (e.g. backpropagation). However, process of optimal choice of hyperparameters in DQN network optimized by search agents is many cases unpredictable and slow. In this paper, we are using genetic algorithms (GA) as search heuristics for new agents introduced in the training process of DQN. We show that DQN + GA approach yields faster convergence to acceptable solution and therefore has potential to be used in practice.**

I. Introduction

Reinforcement learning is the process of dynamically learning by adjusting actions based on continuous feedback to maximize a reward, and it has been widely used in domains where simulated data is available, like games and robotics.

DQN model is used to leverage advances in deep learning to learn policies from high dimensional sensory input [1].

Having that in mind, genetic algorithm search heuristics are implemented as an extra layer on top of the DQN to find the best generated agents for new training episodes. With this strategy, aim is to selectively pick the best performing agent which was the most successful on the given problem. With this approach we can reduce the overall time of training and ultimately converge to the best derived replay solution of our problem. In this project, we have used Cart pole model from OpenAI Gym. Besides aforementioned benefits, genetic algorithm introduces new parameters (i.e. population, crossover function, mutation rate) which can be optimized to optimize training of the new model.

The paper is structured as follows. In the section II we present our learning model, defined problem and comprehensive overview of the carried experiments. Section III contains obtained results, observations and process insights. In section IV we conclude with overview of the obtained results.

II. Experiment

For the purpose of this paper we use DQN with Cart Pole model from Open AI Gym python library. Task is considered "solved" when the agent obtains an average reward of at least 195.0 over 100 consecutive episodes. 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. The episode ends when the pole is more than $15°$ from vertical, or the cart moves more than 2.4 units from the center.

Different reward functions were used throughout the experiment, to analyze the performance of the two approaches in different scenarios. Reward of $+1$ for each time step that the cart remains upright was used in the first scenario, and we argue that this is the easiest policy to learn. It is also prone to locally optimal solutions, e.g. applying a $+1$ force until it reaches the border of the environment, which was shown to give a score of approximately 150. In the second reward scenario, cart was additionally penalized for every change in position compared to its initial position, thus enforcing the cart to stay in place while balancing the pole. Third reward and the most difficult policy to learn, which adds to a reward in first scenario an additional reward for getting to a new position specified as a parameter. This means that the cart needs to get to a new position from the initial while also balancing the pole.

Genetic algorithm implementation can also be considered problem-specific. Since the agents in this case are actually different DQN models, it is appropriate to consider the weights of the DQN network as a genome. Best agents are selected based on the reward they obtained in the final episode of their training. Crossover is then performed between the best agents by exchanging the weights of the agents. Experiments have shown that the best results are obtained when the intermediate recombination is used. There is also a non-zero probability that the new agent will have his weights changed by a sample from uniform distribution on a $[-1, 1]$ interval, which represents mutation.

Two main questions that need answers, are whether genetic algorithms can be utilized to deliver convergence to possibly optimal solution, and, if so, do they converge faster than standard methods. First question can be answered by just observing the reward change of the fittest agent through generations, and the second can be answered by comparing the total execution time and the reward obtained through that time between the approaches. Additionally, we wanted to see how the hyperparameters of the genetic algorithm implementation affect the convergence.

III. Results

In this section, we will present the obtained results. In order to get representative results, each experimental scenario was repeated 20 times and the results were averaged.

Convergence results and the comparison of execution time for the second reward scenario is shown in Figure 1.

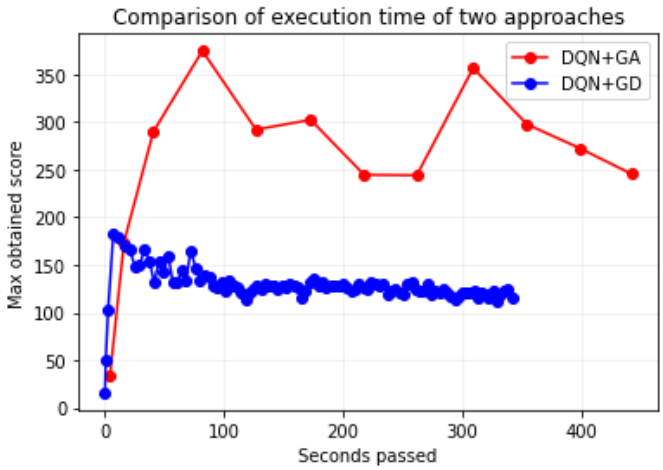

Fig. 1: Comparison of execution time between GA and GD approaches. Score after each episode was taken for GD approach, while the score of the best agent after generation was taken for GA approach.

We can see that the GA approach indeed obtains a better solution compared to gradient descent, which gets stuck in a local optimum. It is worth to note that both approaches, after an initial spike, do not increase their score monotonically, which might indicate an overfitting problem. The two approaches were compared using two other reward policies as well, and the results were very similar to the one presented here.

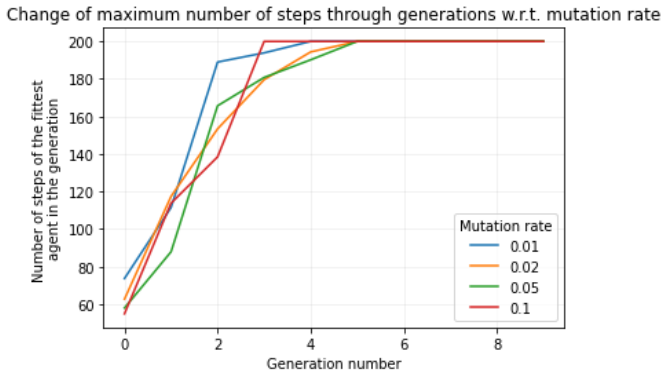

Fig. 2: Simulation results for the network.

Next up, we checked whether the hyperparameters of the genetic algorithm approach can be modified to speed up the convergence. The effect of mutation rate on convergence rate is shown in the Figure 2. We can see that, with a low mutation rate, we achieve on average a very high score already after 2^{nd} generation. While we get max score with each tested mutation rate surely after 5 generations in this experiment, valuable time can be saved by choosing an appropriate parameter value.

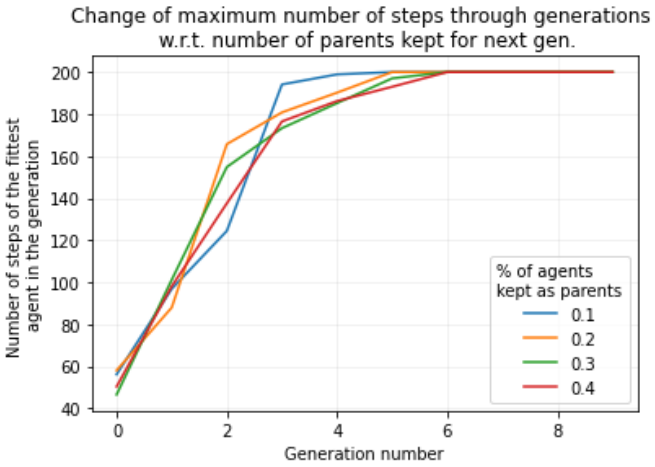

Fig. 3: Simulation results for the network.

We also showed that, by lowering the proportion of agents that are chosen as parents for the next generation, we achieve better results, shown in the Figure 3. With only 10% of best performing agents kept for next generation, we achieved on average highest possible score already after 3 generations.

The results shown here should be taken with a slight dose of restraint. Since this is a probabilistic approach, there are many sources of variability that can be reflected on the end result. In order to get trusted verified results, the experiments should be ran a lot more than 20 times, so we also get a good estimate of uncertainty present in the results. However, this is computationally demanding, and was not conducted in this project.

IV. Conclusion

In the Section III detailed overview of experiment results is given. We can argue that these results still include significant dose of uncertainty, but on the small test sample with given problem, we were able to reduce general training time and improve model results. This was utilized by combining fittest DQN agent to create fitter overall set of agents in a given network. With this in mind, we can argue that numerous hyperparameters are challenge to fit on larger problems due to exhausting and slow hypeparameters fit process. This can take more time from original approach, but we can conclusively say that we improved overall learning process by expanding default training procedure of DQN agents on the given problem.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602