A Comparison of Algorithms for Solving the CartPole Balancing Problem

CS7IS2 Project 2020/2021

Claire Farrell, Daniel Farrell, Joshua Cassidy, Matteo Bresciani

Trinity College Dublin, Dublin, Ireland
{farrec26, farred18, cassidj3, bresciam} @ tcd.ie

Abstract. The growing popularity of reinforcement learning has led to an over-representation in the research of control problems, greatly overshadowing alternative algorithms. Cart-pole balancing is one such control problem. In recent years there have not been many research papers setting out to compare a wide range of algorithms in order to solve this problem, with the majority of research focusing on reinforcement learning approaches. In this research paper, we compare three vastly different algorithms to demonstrate that the cart-pole problem can be solved using a multitude of methods that can perform just as well as reinforcement learning. To do this, we implement a Deep Q-Network, a Genetic Algorithm, and a Proportional-Integral-Derivative (PID) controller. It was found that all approaches solved the cart-pole problem in under 10 episodes, outperforming many previous research attempts. The PID controller and genetic algorithm both outperform the reinforcement-learning-based DQN in terms of the episodes taken to converge and the time taken to do so.

Keywords: Artificial Intelligence, Reinforcement Learning, Cart-Pole, Genetic Algorithm, PID Controller, Gradient Descent, Inverted Pendulum

1 Introduction

The Cart-Pole balancing system is a classic control problem in the field of artificial intelligence. The system has been widely studied in the context of reinforcement learning, but little studies have set out to compare a range of different control techniques to solve the problem. Due to its recent successes, such as with AlphaGo[1] and AlphaZero[2], reinforcement learning has become extremely popular and is seen as a go to choice when dealing with dynamic environments such as Cart-Pole.

The motivation behind this research is to demonstrate that not only can the Cart-Pole system be solved using standard reinforcement learning techniques, but a whole range of algorithms from different families can perform just as well - if not better. This research will compare and contrast three vastly different algorithms for solving the Cart-Pole problem, namely, a Deep Q-Network with experience replay, a Genetic Algorithm, and a Proportional-Integral-Derivative (PID) controller with gradient descent and multiple restarts.

Fig. 1. Cart-Pole in a balanced state vs an imbalanced state

The Cart-Pole problem, also known as the inverted pendulum, consists of a cart that moves along a friction-less horizontal track, along with a pole attached to the cart body that has a centre of gravity directly above its axis. This pole, starting in an upright position, can swing freely around its axis on the cart's body. The goal is to apply appropriate forces to the cart (left or right) in order to keep the pole balanced vertically and to prevent it from falling over. In Figure 1 above, we can see the cart-pole in a balanced state and an imbalanced state.

2 Related Work

As aforementioned, much of the focus of solving the Cart-Pole problem is with using reinforcement learning techniques. Kumar[5], presents research on various types of Reinforcements Learning Algorithms to solve this problem. The algorithms implemented were: Q-Learning, Deep Q-Networks, Double Deep Q-Networks, and Dueling Architectures. The results of this research are that Deep Q-Networks performed, inherently, better than Q-Learning, but that Double Deep Q-Networks, and Dueling Architectures did not provide much improvement than DQN. Results were improved again through the introduction of a experience relay, and thus is the reason we chose this as our first algorithm.

Miranda et al.[6], compared Deep Learning, Q-Learning, and Genetic Programming to solve this problem, finding that Genetic Programming can perform as well, if not better, than Deep Learning models. Other techniques have been compared and surveyed[4], where Genetic Algorithms, BOXES, and an Adaptive Heuristic Critic were implemented in a contained environment. Results concluded that the Genetic Algorithm stabilised faster, and thus, was the reason we chose this as our second algorithm.

Lam et al.[3], solve the Cart-Pole problem using PID controllers where they optimise the parameters using Q-Learning, coined QPID. QPID's results indicate much improvement in solving the problem both from varying initial states and stabilising the pole faster. There is not too much research with Cart-Pole using PID controllers which select the PID parameters using gradient descent, but we can see from [ref] that a common algorithm for choosing the parameters from a PID controller is gradient descent, thus gradient descent with multiple restarts was used. Gradient descent using 10 random restarts was used as we see that gradient descent with 10 random restarts is a very common choice, thus, 10 random restarts were used [ref].

3 Problem Definition and Algorithm

In order to evaluate the chosen algorithms, OpenAI Gym is used to simulate the Cart-Pole problem[7]. As highlighted in the introduction, the goal of the system is to apply appropriate forces to the cart (left or right) such that the pole will remain balanced in an upright position above the cart without falling over.

At each episode, you can observe the following four states:

- The position of the cart
- The velocity of the cart
- The angle of the pole
- The angular velocity of the pole

At any state, the cart only has two possible actions:

- Apply a force of +1 (move the cart right)
- Apply a force of -1 (move the cart left)

The agent will receive a reward of +1 for every episode that the pole remains upright. And the episode will terminate if the pole is more than 20 degrees away from its vertical position, or if the cart moves more than 2.4 units away from its starting location. In Figure 2 below we can see the initial state of the environment and the two possible actions that can be taken.

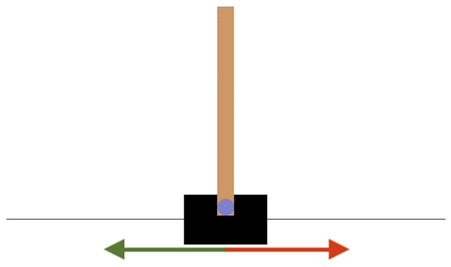

Fig. 2. Cart-Pole initial state and possible actions

3.1 Deep Q-Network with Experience Replay

The Deep Q-Network (DQN) algorithm was developed by DeepMind in 2015 and builds on the well-known Q-Learning algorithm by utilising the power of deep neural networks[8]. Q-Learning is a reinforcement learning technique that strives to find the best action to take given the current state of an environment. It does this by calculating the expected reward for each action at a given step, called a Q-value, and stores this information into a table. This table essentially acts as a “cheat-sheet” for an agent to look up the best actions they should take in the environment. Keeping track of all possible combinations of actions, states, and Q-values, is extremely memory intensive. For example, if an environment had 10,000 states and 1,000 actions per state, this would create a table of 10 million cells. Hence, the people at DeepMind proposed the DQN, which works by approximating these Q-values with the use of a neural network. Using this, there is now no need to store a lookup table and is therefore much more suitable for environments with a large number of states and actions.

Shown below in Figure 3 is a diagram of how the DQN works in the Cartpole environment. The current state St and reward Rt are recorded from the environment and are passed into the input layer of the DQN. As mentioned, there are four states for the Cartpole environment, meaning St will be the position of the cart, the velocity of the cart, the angle of the pole, and the angular velocity of the pole. The DQN then approximates the Q-value for each of the possible actions (move left or move right) that the agent can take given the input state, and the action that produces the largest Q-value is outputted. The agent then takes this action with the highest corresponding Q-value, and the new state and reward are recorded. This process then repeats until convergence. For the Cartpole environment, the algorithm is said to be converged once the average reward is ≥ 195 after episode 100. The reward received is $+1$ for every timestamp that the pole remains upright.

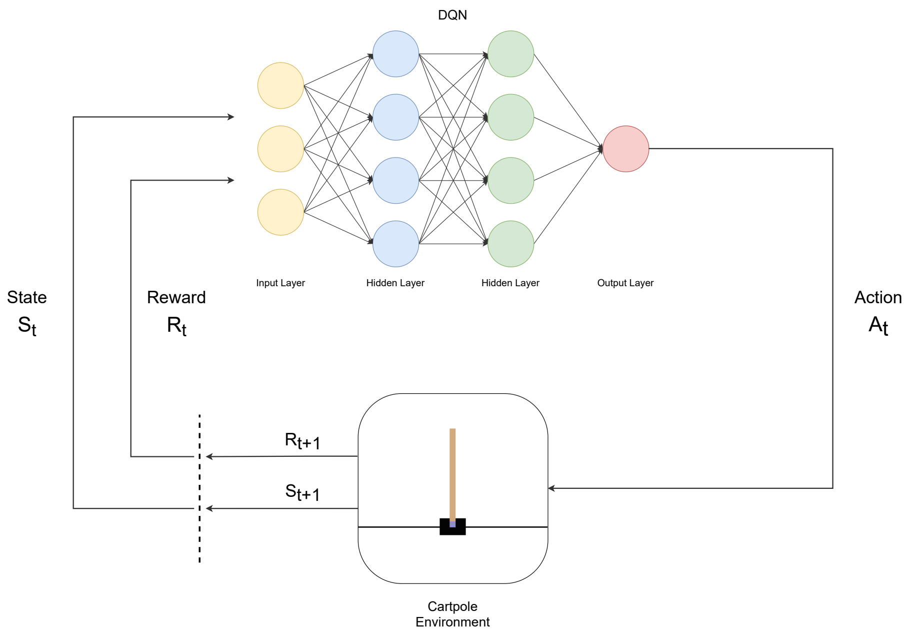

Fig. 3. DQN Diagram

But how does the DQN calculate these Q-values ? Similar to standard Q-Learning the DQN uses a derivation of the Bellman equation for computing Q-values. The exact equation can be seen below.

$$Q^{new}(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Where $Q^{new}(S_t, A_t)$ is the new Q-value, $Q(S_t, A_t)$ is the old Q-value, α is the learning rate, R_{t+1} is the reward of taking the action, γ is the discount rate, and $\max_a Q(S_{t+1}, a)$ is an estimate of the optimal future value.

3.2 Genetic Algorithm

The Genetic Algorithm is a metaheuristic algorithm that is inspired by natural selection. It makes use of a population of individuals. An individual is a possible solution to the problem and is composed of chromosomes that represent all the properties required to solve the problem.

In our case, the population is composed of individuals with 4 chromosomes:

- The position of the cart
- The velocity of the cart
- The angle of the pole
- The angular velocity of the pole

The first step of the algorithm consists of initializing the population, in our case we did that by giving random values to the 4 parameters for every individual in our population. The next steps of the algorithm are repeated until convergence:

- Selection
- Crossover
- Mutation
- Elitism

In the selection step of the algorithm, we apply a fitness function to calculate the scores of each individual in our population. Based on this score, we will know which individuals are fitter than others and we will choose the ones with a greater score to be the parents. To choose the parents, we select a group of individuals(two or more) and we will select two parents from that group(the fittest individuals in the selection group).

Once the parents are selected, we will go to the crossover phase. In this phase, the chromosomes of the parents will be combined, so to get new individuals that will go to the next generation. In our algorithm, we do that in two possible ways. The first one is to compute the average for each chromosome from both parents. In the second one, we just take two chromosomes from one parent and two from the other one.

The last step is called mutation. In this step, each chromosome of each individual generated from the crossover has a certain probability of mutating. When the mutation happens, the value of that chromosome is increased or decreased by a random amount. Some individuals though, already have a very high fitness score and, on top of being chosen as parents, will also go to the next generation unchanged. This step is called elitism. We take the fittest individuals of this generation and we pass them to the next one.

3.3 PID Controller using Gradient Descent with Random Restarts

PID controllers are widely utilised due to their high performance and overall robustness in uncertainty, like regulating temperature. It uses a feedback loop that controls the three parameters by tweaking the values in accordance to the error. When optimising this controller, there are three parameters to be tuned. The parameters are proportional gain (P), integral time (I), and derivative time (D)[9].

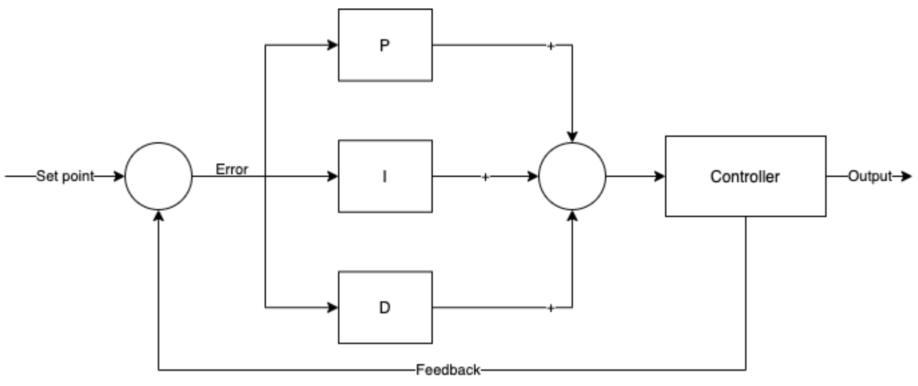

Fig. 4. PID Controller Diagram

The PID controller using Gradient Descent was implemented by using a learning rate of 0.05 and 10 random restarts selecting the initial P, I, and D values to be value between zero and one. Gradient Descent is used to optimises the P, I and D values by minimising the error of the position of cart, velocity of cart, angle of pole respectively. The error in these values is computed by the difference of the goal state values and the terminating state values. Gradient Descent is run for 200 iterations and the PID controller is trained in the environment for 110 episodes.

Algorithm 1: PID Controller with Gradient Descent

```
Result: Write here the result
initialization;
while While condition do
    instructions;
    if condition then
        instructions1;
        instructions2;
    else
        instructions3;
    end
end
```

3.4 Baseline Model

In order to adequately evaluate the algorithms, a baseline agent by creating an agent that navigates the search space by randomly choosing an action at each state. This model was evaluated and the results were used to understand the performance of the algorithms implemented.

4 Experimental Results

The three algorithms were each implemented in the same environment and under the same conditions to ensure they could be evaluated effectively. Each of the environments used a random seed of 7 and each of the algorithms being run for 200 episodes. As aforementioned, the Open-AI gym simulated the Cart-Pole environment and the following metrics were captured to evaluate the algorithms: Final Reward, Number of episodes, Average Reward, and Time Taken in Minutes.

Evaluation Metrics				
Algorithm	Final Reward	Number of episodes	Average Reward	Time Taken (Minutes)
Baseline	40.0	Does not converge	34.75	Does not converge
DQN with Experience Replays	500.0	6	199.14	12.18
Genetic Algorithm	500.0	1	199.13	0.022
PID Controller with Gradient Descent	500.0	0	500.0	1.30

Table 1. Performance results of algorithms

Table 1 shows the performance metrics of each of the algorithms in the Cartpole environment, from this table we can see that the each of the implemented algorithms far outperform the baseline model which only manages to achieve a reward of 40, with the other algorithms achieving a reward of 500 which is the maximum reward that is achievable in this environment. We can see that the best performing algorithm in terms of time take to complete running is the Genetic algorithms agent which only takes 3 seconds to converge which is quicker than the Baseline agent and the other agents making the genetic algorithms agent the best in terms of time taken to train. The baseline takes only 4 episodes to converge making it the quickest to converge in terms of episodes but this is because the algorithm does not learn or have a high reward. We can see that the DQN with experience replays takes the longest to train taking approximately 25 and a half minutes complete this is due to the number model taking a long time to learn and being a computationally expensive algorithm to run, this makes this algorithm the poorest in the time taken to train.

4.1 Results Discussion

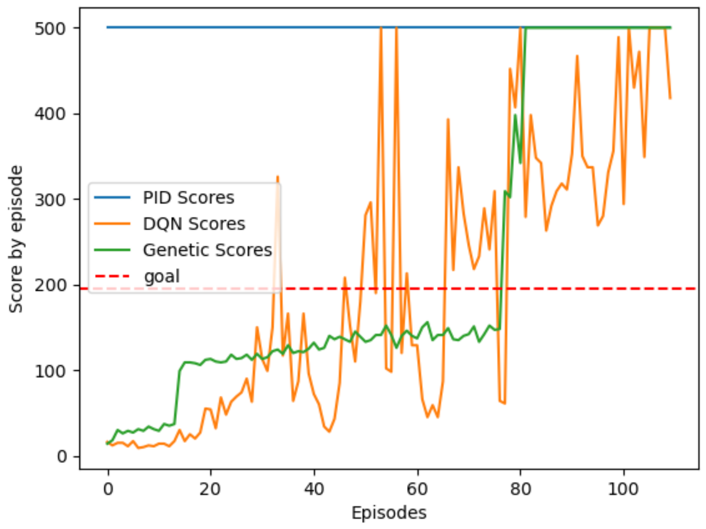

Fig. 5. Cart-Pole Compound Plot

To further compare the behaviours of each agent, they were left to run for 110 episodes. Figure 5 above shows the rewards gained by each of the agents against the episodes that have elapsed. The blue line represents the PID agent which we can see constantly achieves a maximum reward of 500 for the entirety of the 110 episodes. The green line represents the best agent in each of the populations throughout the training of the genetic algorithm. As we can see, the reward gained by the genetic algorithm gradually increases as the episodes elapse, and at around episode 80 it steadily achieves the maximum reward of 500. Finally, the orange line depicts the rewards achieved by the DQN agent. As shown, the rewards achieved by the DQN do increase with the episodes elapsed, but they tend to fluctuate drastically. The PID controller and genetic algorithm learn at a much steadier rate.

As mentioned previously, the Cart-Pole convergence condition is defined as being when the agent achieves an average reward of ≥ 195 after 100 episodes. The initial 100 episodes are seen as a training period for all algorithms. As depicted in the results table previously, the PID Controller converged in 0 episodes which we can see why in the above plot, since it constantly achieved a reward of 500. The genetic algorithm converged on episode 1 (i.e. 100 training episodes + 1), and the DQN converged on episode 6.

5 Conclusions

From the results highlighted in the previous chapter, it is clear that complex reinforcement learning methods are not always necessary when working with control problems like Cart-Pole. Alternative methods such as the use of Genetic Algorithms or PID Controllers can actually provide similar results and do not require as much time or processing power as a Deep Q-Network. For this reason, this research has succeeded in what it set out to do, by demonstrating that alternative, somewhat nontraditional solutions do exist for the Cart-Pole control problem.

Building upon this research, there are some opportunities for further improvements. For example, the Genetic Algorithm may be improved with finer hyper-parameter tuning. Exploring different values for the population size, the mutation chance for every chromosome, and the number of top individuals that will directly go to the next generation is something that would help the algorithm converge in a more rapid way. Also, for the PID Controller, different optimisation algorithms can be tested besides gradient descent. For example, heuristic tuning methods such as Ziegler-Nichols or Particle Swarm Optimisation can be used.

References

1. Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016). doi.org/10.1038/nature16961
2. Silver, D. et al. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. ArXiv (2017). abs/1712.01815
3. Lam, Hak-Keung Shi, Qian Xiao, Bo Tsai, Shun-Hung. (2018). Adaptive PID Controller Based on Q-learning Algorithm. CAAI Transactions on Intelligence Technology. 3. 10.1049/trit.2018.1007
4. M. Randall, C. Thorne and C. Wild, "A standard comparison of adaptive controllers to solve the cart pole problem," Proceedings of ANZIIS '94 - Australian New Zealnd Intelligent Information Systems Conference, 1994, pp. 61-65, . doi:10.1109/ANZIIS.1994.396949
5. Kumar, S. (2020). Balancing a CartPole System with Reinforcement Learning–A Tutorial. arXiv preprint arXiv:2006.04938.
6. Í. M. Miranda, M. Ladeira and C. de Castro Aranha, "A Comparison Study Between Deep Learning and Genetic Programming Application in Cart Pole Balancing Problem," 2018 IEEE Congress on Evolutionary Computation (CEC), 2018, pp. 1-7, doi: 10.1109/CEC.2018.8477814.
7. OpenAI Gym. (2021). A toolkit for developing and comparing reinforcement learning algorithms [Online]. Available at: https://gym.openai.com/[8 April 2021]
8. Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). doi.org/10.1038/nature14236
9. H. Wu, W. Su and Z. Liu, "PID controllers: Design and tuning methods," 2014 9th IEEE Conference on Industrial Electronics and Applications, Hangzhou, China, 2014, pp. 808-813, 10.1109/ICIEA.2014.6931273