# Application of Reinforcement Learning Algorithms to the Cart Pole Balancing Problem

Eric J. Kim

# Contents



# List of Figures



# List of Tables

# 1 Project Overview

The objective of this project was to replicate the reinforcement learning (RL) algorithms implemented in the paper "Comparison of Reinforcement Learning Algorithms applied to the Cart-Pole Problem" by Nagendra et al., published in IEEE from the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI) [1].

As such, the following algorithms were implemented as control solutions to the cart pole balancing problem:

1. Q-Learning Off-Policy Temporal Difference
2. Actor-Critic Policy-Gradient
3. Temporal Difference with Value-Function Approximation

This report will describe the details and procedures taken to implement these algorithms, examine their resulting policies, and state what was learned from this project.

# 2 Cart Pole Balancing Problem

The cart pole balancing task is a variant of the classical inverted pendulum nonlinear control problem. The cart is modelled as a rigid body able to move along a one-directional line, with one end of a rigid pole fixed to the centre of the cart at a pivot point (Figure 1). The objective of the controller is to move the cart back and forth along a single dimension to balance the pole at the unstable equilibrium $\theta = 0$, where $\theta$ is the angle off the vertical axis, positive clockwise.

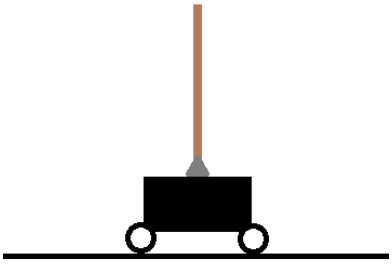


Figure 1: Cart Pole Physical Model

The system dynamics were modelled with the differential equations (Equations 1 and 2), as detailed in the paper [1]. The system parameter values, which were kept constant across all simulations, are listed in Table 1.

$$\ddot{\theta} = \frac{(M+m)gsin\theta - cos\theta(F + ml\dot{\theta}^2 sin\theta)}{\frac{4}{3}(M+m)l - mlcos^2\theta} \tag{1}$$

$$\ddot{x} = \frac{F + ml(\dot{\theta}^2 sin\theta - \ddot{\theta}cos\theta)}{M+m} \tag{2}$$

$M$ = mass of cart
$m$ = mass of pole
$g$ = acceleration due to gravity
$l$ = distance from center of mass to pivot

The objective of an agent is to keep the pole angle within 12° from the vertical and the cart position within a 4.8m long track. The environment is deterministic, so taking action $a$ from state $s$ results in a transition to state

Table 1: Simulation Parameters

| Parameter | Value | |
|---|---|---|
| $M$ | 0.711 | kg |
| $m$ | 0.209 | kg |
| $g$ | 9.8 | $m/s^2$ |
| $l$ | 0.326 | m |
| $\tau$ (simulation sampling interval) | 0.02 | s |

$s'$ with a probability of 1. Additionally, the initial state is randomized to be between the defined state boundaries. For evaluations using discrete state spaces, the state space was discretized into two different granularities shown in Table 2. Both discretizations are identical apart from the increased number of bins for the pole angle $\theta$.

Table 2: Discretizations
(m, m/s, degrees, degrees/s)

| State | Discretization 1 (low) |
|---|---|
| $x$ | $[-2.4, -0.8), [-0.8, 0.8], (0.8, 2.4]$ |
| $\dot{x}$ | $(-\infty, -0.5), [-0.5, 0.5], (0.5, \infty)$ |
| $\theta$ | $[-12, -6), [-6, -1), [-1, 0), [0, 1), [1, 6), [6, 12]$ |
| $\dot{\theta}$ | $(-\infty, -50), [-50, 50], (50, \infty)$ |
| | Discretization 2 (high) |
| $\theta$ | $[-12, -6), [-6, -4), [-4, -3), [-3, -2), [-2, -1), [-1, 0),$ $[0, 1), [1, 2), [2, 3), [3, 4), [4, 6), [6, 12]$ |

For each of the algorithms, learning was halted when the policy resulted in the agent keeping the pole upright and balanced for the entirety of the simulation interval, which was set to 200s. Lastly, the tkinter graphics library was used to create the animations of the resulting policies.

# 3 Q-learning: Off-Policy TD

The q-learning algorithm that was implemented in the paper and this project used a tabular method of storing state-action values. As such, both discretizations outlined above were tested. Furthermore, the reward structure for the q-learning algorithm penalized by $-1$ when an agent left the state bounds and rewarded 0 for every time step in which the pole was balanced and the cart remained within its track limits. This reward method is valid due to the discount factor $\gamma$, which effectively reduces the penalty for agents that keep the pole upright for longer. The implementation of the algorithm was based on the pseudocode provided in the paper and was ultimately trivial to code and make functional. Tables 3 and 4 show the results of the project (exploration factor set to 1e-4) and paper respectively.

Table 3: Q-learning Project Results ($\epsilon$ = 1e-4)

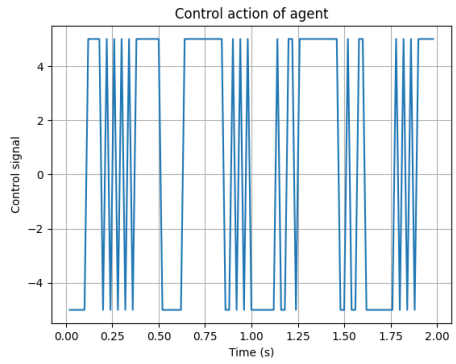
| Force (N) | Learning rate ($\alpha$) | Discount factor ($\gamma$) | Discretization | Theta (°) | x (m) | Episodes to balance |
|---|---|---|---|---|---|---|
| ±10 | 0.5 | 0.99 | Low | [-9.7, 9.6] | [0.6, 2.4] | 1151 |
| ±15 | 0.6 | 0.99 | High | [-8.6, 9.6] | [-1.0, 1.5] | 852 |
| ±30 | 0.4 | 0.99 | Low | [-6.1, 5.6] | [-0.86, 0.76] | 333 |

Table 4: Q-learning Paper Results [1]

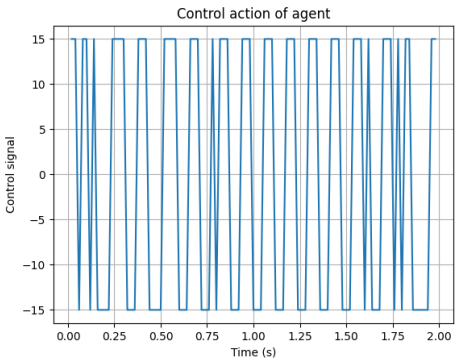
| Force (N) | Learning rate ($\alpha$) | Discount factor ($\gamma$) | Discretization | Theta (°) | x (m) | Episodes to balance |
|---|---|---|---|---|---|---|
| ±10 | 0.5 | 0.99 | Low | [-11, 10] | [-1.5, 2.4] | 420 |
| ±15 | 0.6 | 0.99 | High | [-2.9, 2.9] | [-1, 1] | 380 |
| ±30 | 0.4 | 0.99 | Low | [-3, 6.3] | [-0.1, 0.6] | 24 |

The results presented above show that the q-learning implementation yielded similar trends to those of the paper. Increasing the force with which the cart is moved by selecting one of the two actions (left or right) reduced the number of episodes required before the agent could balance the pole for the full simulation time. This is primarily due to the value of the simulation time step $\tau$. The time step of 0.02 was found to be small enough that the agent could rapidly alternate between pushing the cart left and right, even with a greater action force. Figure 2 shows the control signals of the q-learning agent for 4 different force values. It is clear that higher values result in more a rapid alternating of the left and right control signals, which allows for faster stabilization of the pole. However, such a system would most likely not be feasible in a physical cart pole system due to inevitable control signal delay and the non-trivial transient period during which a motor would accelerate to produce a force.

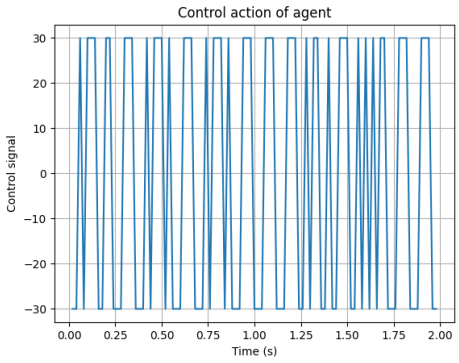
Figure 2: Control Signals of Various Action Forces



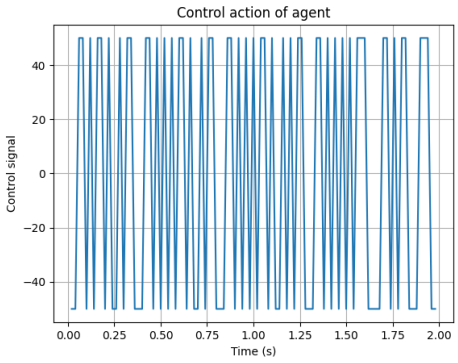

(a) Force = 5 N




(b) Force = 15 N




(c) Force = 30 N




(d) Force = 50 N

In addition to discussing the q-learning algorithm in the context of the cart pole problem, further evaluations were conducted to examine the behaviour of the algorithm with respect to its hyperparameters. Tables 5 and 6 show the results of varying the exploration factor $\epsilon$ and discretization respectively. The discount factor $\gamma$ was set to 0.99 for a relatively long time horizon, since the goal is to keep the pole balanced for as long as possible.

Table 5: Q-learning $\epsilon$ Analysis ($\gamma = 0.99$)

| Force (N) | Learning rate ($\alpha$) | Exploration factor ($\epsilon$) | Discretization | Theta (°) | x (m) | Episodes to balance |
|---|---|---|---|---|---|---|
| ±15 | 0.6 | 1e-3 | Low | [-8.4, 10.5] | [-1.9, 0.54] | 10844 |
| ±15 | 0.6 | 1e-6 | Low | [-11.6, 8.6] | [-0.20, 1.2] | 1177 |
| ±15 | 0.6 | 1e-9 | Low | [-4.0, 4.0] | [0.044, 0.66] | 559 |
| ±15 | 0.6 | 1e-12 | Low | [-8.1, 7.2] | [0.47, 0.99] | 3147 |

Table 6: Q-learning Discretization Analysis ($\gamma = 0.99$)

| Force (N) | Learning rate ($\alpha$) | Exploration factor ($\epsilon$) | Discretization | Theta (°) | x (m) | Episodes to balance |
|---|---|---|---|---|---|---|
| ±15 | 0.1 | 1e-9 | Low | [-8.4, 10.5] | [-1.9, 0.54] | 1262 |
| ±15 | 0.1 | 1e-9 | High | [-6.1, 4.0] | [-0.70, 0.85] | 17168 |

The results show that reducing the exploration factor also reduced the number of episodes required before convergence to a successful solution, until an inflection point is reached. An exploration factor $\epsilon$ of 1e-9 resulted in the lowest number of episodes required, while an even lower value of 1e-12 increased the number of episodes. Unsurprisingly, due to the low numbers of states and actions, less exploration was favoured. Only a small number of state-action pairs were worth exploring, as many would result in the pole falling over. However, exploration values that were too small still prevented the finding of the optimal actions. It was also noted that changing the discretization by increasing the number of bins for the state $\theta$ resulted in an increase in episodes before a solution was found, because of the larger state-action space that needed to be explored. An animation of a successful q-learning policy can be found in the project directory.

# 4 Actor-Critic Policy-Gradient

The actor critic algorithm was more difficult to implement than q-learning, both in terms of following the procedures of the paper and successfully using neural network function approximators. Specifically, various network parameters such as the way the policy was encoded in the actor network and the precise neuron activation functions used had not been included in the paper, and as a result, there exists slight discrepancies between the project and the paper in terms of implementation.

The process that resulted in the successful actor critic implementation began by attempting to directly implement the algorithm provided in the paper. The authors stated that both the actor and critic networks each consisted of a single neuron. As such, it was assumed that the activation function for this network was linear, and consequently, the critic network had simply been a linear function approximator. However, the method through which the actor network provided a policy parameterization in the form of probabilities was unclear. In the case of the cart pole problem, which has a total of two possible actions, the actor must output two probabilities, one for each action. This cannot explicitly be done with a single neuron, unless this single output represents a probability of taking one particular action, bounded between 0 and 1 through the use of a bounding function such as the sigmoid function. Regardless, the method through which this was achieved was not specified. After encountering these difficulties, it was decided to implement the actor critic algorithm with two neural networks, each with a hidden layer containing more than one neuron.

The pseudocode reflecting the Python implementation is as follows:

```
Create and initialize actor and critic networks with random weights
for each episode:
     Fetch initial state S
     for each step in episode:
          Sample action A from actor network
          Obtain next state S', reward R
          Compute neural network update targets
          Update network weights using the targets and the inputs
          S ← S'
     end
end
```

The primary challenge for this multi-neuron implementation, which was created using the python Keras library, was tuning the actor and critic learning rates. Typically, learning rates that are too large would be expected to overshoot the target update values, and rates that are too small would prolong the already lengthy process of training, which involves computing gradients. Nevertheless, the actor critic network was able to train successful policies with learning rates in the order of 1e-3 all the way to 1e2. Unsurprisingly, higher learning rates for both the actor and critic resulted in less episodes before a successful policy was found. This result of being able to use relatively high learning rates is also noted in the paper, and is believed to be a consequence of the simplicity of the two-action control problem. In addition, it was learned that the critic network should have a learning rate that is around 5 times larger than that of the actor. If the actor learns faster than the critic, the values with which the actor network weights are updated would be inaccurate, since they would be based on the outdated outputs of the critic network.

The reward function for this algorithm was defined by rewarding a value of 1 at every time step for which the cart and pole stayed within the bounds, and a reward of 0 for exceeding the limits. Regarding the neural network approximators, it was found that normalizing the continuous states using the state boundaries facilitated learning, since the network inputs were on the same scale, consequently reducing the chances of developing extreme differences in the weight parameters.

Although a hyperparameter analysis for the actor critic algorithm similar to that conducted for q-learning was not practical due to the lengthy computational times of the neural network libraries, the implemented algorithm was able to train policies that kept the pole balanced for the simulation time, which was reduced to keep the runtime reasonable. An example of a run limited to 300 episodes is shown in Figure 3.

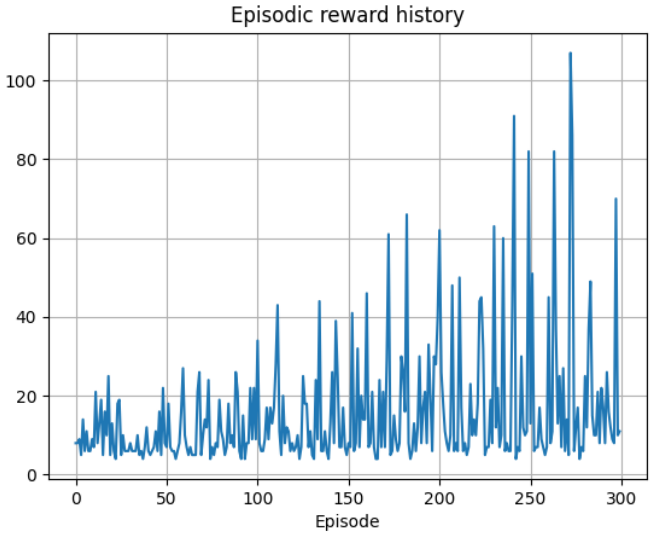



Figure 3: Actor Critic Learning

# 5 TD with Value Function Approximation

The temporal difference (TD) with value function approximation was implemented based on the actor critic algorithm's use of neural networks. This reinforcement learning method used a linear function approximation for the state-value function. The pseudocode provided in the paper resembled a SARSA algorithm, as the TD error explicitly used the next state and action, as shown below.

$$\Delta w = \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w) - \hat{q}(S_t, A_t, w))\phi(S_t, A_t) \tag{3}$$

However, there was no mention of an exploratory mechanic such as the epsilon-greedy mechanism, or details on the feature vector selection. As such, the decision was made to incorporates epsilon-greedy action selection and create a feature vector composed of the cart pole system states concatenated with the action that is selected.

The pseudocode reflecting the Python implementation is as follows:

```
Create and initialize action-value approximation function with random weights
for each episode:
     Fetch initial state S
     Fetch initial action A using epsilon-greedy mechanism
     for each step in episode:
          Obtain next state S', reward R
          Fetch next action A' using epsilon-greedy mechanism
          Update function weights based on Equation 3
          S ← S'
          A ← A'
     end
end
```

A range of values for both the learning rate and the exploration factor $\epsilon$ were tested, spanning from 1e-3 to 1e3 and 1e-3 to 1 respectively. However, the algorithm failed to produce a successful policy even after 100000 episodes. It was initially believed that the constant learning rate and exploration factors were preventing the algorithm from learning, and as such, both of these values were set to gradually decay. Through testing, it was found that having large $\epsilon$ values resulted in better performance to a certain degree, as shown in Figure 4, which shows the evaluation of 50000 episodes with a decaying learning rate and exploration factor.

The apparent lack of exploration, or the agent's consistent selection of a single action, was believed to be a result of an inadequate feature vector. Therefore, a polynomial-based feature vector was constructed as shown below.

$$\phi(s, a_1) = [x, \theta, \dot{x}, \dot{\theta}, x^2, \theta^2, \dot{x}^2, \dot{\theta}^2, 0, 0, 0, 0, 0, 0, 0, 0, 1] \tag{4}$$

$$\phi(s, a_2) = [0, 0, 0, 0, 0, 0, 0, 0, 0, x, \theta, \dot{x}, \dot{\theta}, x^2, \theta^2, \dot{x}^2, \dot{\theta}^2, 1] \tag{5}$$

The function approximator weights had dimensions of $1 \times 9$, and the feature vector would be constructed depending on the action selected. For instance, vector (4) would be the input to the function approximator if the action were 1, or to push the cart to the right. This was necessary in order to separate the effects of the two actions, since they both would have different action values. Finally, a bias term was added to the feature vector so that the value estimates could be scaled independently of the state features. Testing this function input however, did not produce significant improvements over simply using the cart pole states as features directly.

With the difficulty in determining the cause of such constant action selection, a neural network approach for implementing the temporal difference function approximation algorithm was implemented. This method was based on the actor-critic implementation, using the Keras python library. The neural network approximator

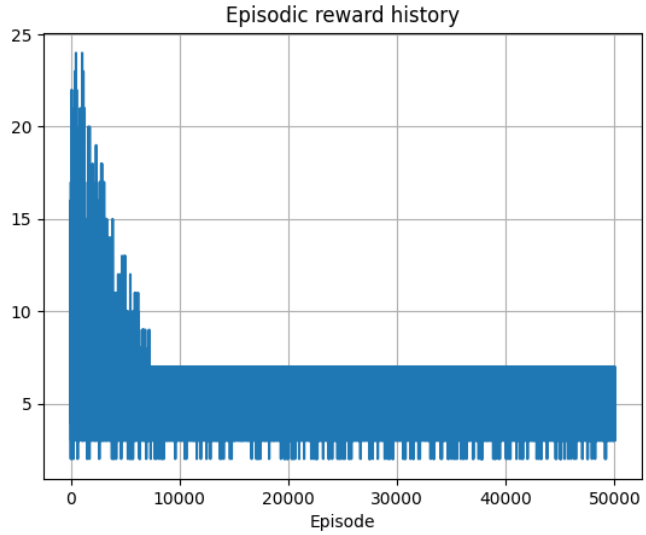



Figure 4: TD decaying $\alpha$, $\epsilon$

consisted of a single hidden layer of 24 neurons, a single output for the value, and an input composed of the cart pole states and the selected action. A nonlinear relu activation function was used, with the network updates based on the mean squared error of the TD error $\delta$. This nonlinear function implementation proved to be more successful than the linear approximator attempts. It is believed that there was insufficient coupling between the states in the linear version of the algorithm, which made learning difficult. As a result, the next logical step would be to implement a radial basis function or a coarse coding mechanism to increase the level of interaction between the input dimensions.

However, despite evidence of the algorithm's converging towards the optimal policy, some evaluations eventually led to a sudden reversion to an unlearned, or elementary, policy. This phenomenon, common to function approximation methods and known as catastrophic forgetting, is illustrated in Figure 5.

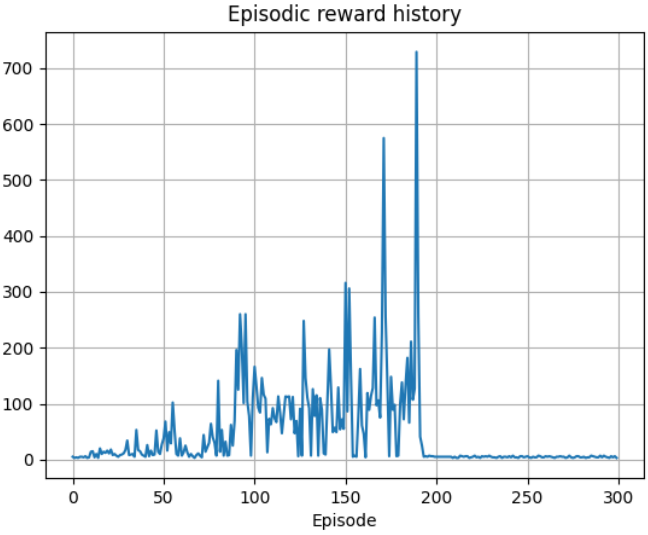



Figure 5: TD catastrophic forgetting

In the context of this project, catastrophic forgetting occurs when a network, who has only explored and

learned a certain portion of the state-action space, suddenly encounters a new area of the domain, with which it is unfamiliar and untrained. Consequently, the network catastrophically fails, as its understanding of the search space, represented by neuron weights, is severely underdeveloped. This relative overfitting of a locally optimal solution is a direct result of using function approximators such as neural networks, and can be mitigated through network regularization and low learning rates.

# 6 Conclusion

The aim of this project was to implement and explore three different reinforcement algorithms in the context of a canonical control problem through replication of a paper. All three algorithms were implemented with varying levels of success, and the majority of the effort was invested in achieving the level of implementation and identifying the behaviours described in the procedures above.

The level of detail concerning the precise implementation of this algorithm in the paper was insufficient, and as a result, it was not possible to accurately replicate the results of the paper. Nevertheless, the aforementioned processes allowed for a greater understanding in function approximation for reinforcement learning. Although function approximation methods may reduce the memory requirements for complex problems when compared with tabular methods, they are also much less straightforward to implement, and often require determining empirically the precise hyperparameter values and network characteristics that result in successful policies while managing unique phenomenon such as catastrophic forgetting.

The conclusions for each of the algorithms are summarized as follows:

**1**. Q-Learning Off-Policy Temporal Difference

- Tabular method is simple to implement
- Restricted to discrete states and actions
- Scales poorly with more complex problems

**2**. Actor-Critic Policy-Gradient

- Neural network approximators require hyperparameter tuning
- Parameter values are not initially clear and are highly problem dependent
- Implementation requires more computational resources to solve for gradients

**3**. Temporal Difference with Value-Function Approximation

- Allows for continuous state-action spaces
- Only requires a single neural network
- Linear function approximation requires specifically curated feature vector

# References

[1] S. Nagendra, N. Podila, R. Ugarakhod, and K. George, "Comparison of reinforcement learning algorithms applied to the cart-pole problem," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, pp. 26–32.