AGRICULTURAL JOURNAL
MEDWELL PUBLICATIONS

# Deep Reinforcement Learning Applied to Cart Pole Game

Abdallah Al-Zu'bi and Ahmad Al-Qerem
*Computer Science PSUT, Amman, Jordan*

**Key words:** Deepmind deep learning, reinforcement learning, cart pole, Markov decision process, Bellman equation

**Corresponding Author:**
Abdallah Al-Zu'bi
*Computer Science PSUT, Amman, Jordan*

Page No.: 137-142
Volume: 15, Issue 6, 2020
ISSN: 1816-9155
Agricultural Journal


**Abstract:** Building an agent to play games might be done in several ways, like mini-max, Monte Carlo tree search, deep learning or it could be a combination of two or three technics or even more, like the popular chess computer engine deep blue or alpha go. This document built an agent that plays and balances cart pole game, the agent used deep reinforcement learning-specifically Q-learning-algorithm to build neural network that greedy to maximize the reward function and balance the pole for the longest period at the end we got an agent that outperform the random and human agent.

## INTRODUCTION

In the era of Artificial intelligence and machine learning, there are a lot of terminologies that flow to the top, one of them is Deepmind or google deepmind, it is a combination of deep neural network and reinforcement learning[1] or you can consider it as reinforcement learning with deep network that approximate the value function or Q value function. We are going to discuss each part in separate section.

Reinforcement learning is an AI algorithm that tries to learn from the environment by taking sequence of actions, then get rewards for these actions, after each action the agent checks the new state of the environment, it tries to maximize the reward function and builds a policy to take the best sequence of actions to maximize the cumulative rewards, the policy can be defined as the mapping between environment state and actions, so given an state to the agent, he can take an action if he has a policy function[2].

Deep learning refers to neural network with some complexity, usually the neural network has more than two hidden layers, Q learning is an example of deep learning and we are going to discuss it in separate section. The results of creating deepmind agent that plays atari game was surprising, a lot of agents surpasses profession human players[3].

Deepmind has a lot of applications like self driving cars, Robotics, healthcare[4], however, it could be extended to any discipline where the agent can learn and take experience from interacting with the environment to build a strategies to take a good sequence of actions or maybe the optimal sequence if you build the neural network and that approximate the Q value function and tune it well.

AlphaGo is one of the most popular deepmind agent, it is the first agent that can play and defeat champion's Go players, it has been announced as the best go player ever in the history, after defeating the world champions Seoul in 2016 four times before it got lost one time[5].

Our agent plays cartpole game and tries to keep it balance as much as possible, the input of the game is image of pixels, we used a game from GYM toolkit, which is open source AI library, it is built for reinforcement learning purpose as it simulates the game environment so you can take an action and get reward or

score for each action then get the new environment state for each action using built in API. There are many games you could test your reinforcement algorithm on them, so it is very helpful specially for beginners and it is used to compare between reinforcement learning algorithms that built to resolve games[6].

## REINFORCEMENT LEARNING

It is defined as the process of building agent that is able to learn from interacting with the environment, takes good or bad consecutive actions and keeps improve himself, it is simulate how human being learns[1], let's say when you are trying learn driving car, here you are the agent and the road, car, traffic lights. represent the environment, the agent keeps take actions and if he drives good he will get a positive reward and if not he will get a punishment or negative reward until reaching the destination, he does this many times to learn how to drive well and follow the policies, the agent tries to maximize the sum of rewards that could be getten each trial by driving well and avoiding driving in a bad way and taking the shortest path. So, the agent keeps taking actions, e.g., take the next right or left and this will change the environment and taking reward or punishment for each action. actually it is not bruteforce process as some people think, as the agent after a while will learn to stop at the red traffic light and pass the green one, limit his speed on some roads, taking the shortest path, even if he doesn't explore all the environment and even if road speed limit has been changed, so, defining the reward functions is one of the most important point in the RI to build a genius agent, some people defined it as an art, it is not necessary to be a complex one but it should be a suitable for the environment, e.g., Some Atari Games uses the score that returns from the game, our agent for cart pole uses the time steps, he can get a punos rewards for each time step he keeps the pole balanced.

As you can see in Fig. 1, the agent keeps do actions and the environment keeps updated, then update the agent with new rewards and states, the agent use this new state to take the next action, so, the agent tries to maximize its rewards[8].

$$\text{Reward} = R_0 + R_1 + R_2 + ,..., + R_N$$

In some situation like self -driving car, the agent will keeps driving and may go in a cyclic tour as this will increase its reward, so, we used to multiply the reward value with discount factor ($\gamma$), so that, the partial you take from the future reward will be less than you take from the current reward in that way, we force the agent to reach the goal state or finish the game faster to maximize its rewards[8].

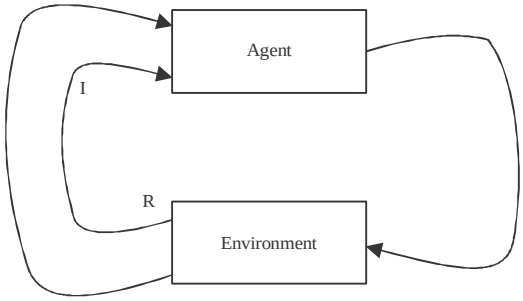


Fig. 1: Reinforcement learning paradigm

$$G_t = R_0 + \gamma R_1 + \gamma^2 R_2 + ,..., = \sum_{i=0}^{\infty} + \gamma t \times + R_{t+1}$$

where, $\gamma$ is hyper-parameter between 0 and 1, to force the agent to finish the game earlier. So, the expected reward for the agent if he follows the policy $\pi$ in the state s will be defined in this equation and this value is called state value function[2]:

$$V_{\Pi_t}(s) = E\pi\left[\sum_0^{\infty} \gamma^k R_{k+t+1} \mid S_t = s\right]$$

Where:
$\Pi$ = Policy
$E\pi$ = Expected value when following the policy

The agent explores the environment and gets experiences from it and tries to build its own policy which is a matrix (plan) that maps state to an action.

There is another term that should be defined which is Q-value or quality value function, it is the reward that the agent expects to have if he takes an action a in state s and then following the policy, so it is defined the quality of the given state a when you take an action a and this can be defined in following math equations[8]:

$$Q_{\Pi_t}(s,a) = E_{\Pi}\left[\gamma^k R_{k+t+1} \mid S_t = s, A_t = a\right]$$

We could rewrite it as[2]:

$$Q_{\pi_t}(s,a) = E_{\Pi}\left[r \mid s,a\right] + \gamma \sum_{S'} P(s' \mid s,a) \times V_{\Pi^*}(s')$$

So, the input of the Q function is a state an action and policy while the output is the expected reward, the agent will try to maximize this value, so he will chose the policy that maximize this value[2].

$$Q_*(s,a) = \max_{\pi} Q_{\pi}(s,a)$$

So, the agent will find the optimal policy while he interacts with the environment and getting reward and punishment. Keeps going between exploration and

exploitation until reaching a converged value for Q-value function to build the policy, this could be done using value iteration or policy iteration.

As you can see from the equations above they use recursive so we need to approximate these values and this is done using deep neural network.

## DEEP LEARNING

It is a class of machine learning algorithm, in which the model consists of neural networks that has more than one hidden layer, emulating the human brains of thinking, this network is trained using training dataset and then used to predict and take a decision for any new entries, it is used in many discipline like approximate the expected agent's reward function in our example, spam and Fraud detection, image classification and in many others fields[7, 8].

## MARKOV DECISION PROCESS

MDP is the mathematical model that is used to represent the sequence of decisions making process in any environment[9], actually it is built during 1950s, it is a generic model widely used in machine learning specifically in reinforcement learning for representing a model that has both parts of decisions making, stochastic and computed decisions, this model tries to optimize and build an agent that has a perfect reaction for the environment states[2]. In this project, MDP is used to build our agent that starts acting randomly then build its own greedy policy or the plan to increase its reward from the environment.

So, the mathematical model contains a set of environment, set states S and for each state the agent have to take an action a from actions set A and this will change the cumulative reward value for the agent then as a consequence of taking an action change the current state s to the next state. so we are talking about trajectory of states actions and reward[1, 2]:

$$S_0, A_0, R_1, S_1, A_1, R_2, ..., S_t, A_t, R_{t+1}$$

## BELLMAN EQUATION

It is a mathematical optimization techniques that divides a dynamic optimality problem to subproblems as it will be easier to resolve these subproblems and then merge them together to resolve the main problem, it is applied to many fields like engineer control theory, economic, applied mathematic, machine learning and algorithm optimization. it is named for Richard E. Bellman who is applied mathematician, the Bellman equation is the spark idea for the dynamic programming[2, 10], in our RI solution we used it to build optimal policy or mapping function for the agent, Bellman equation is build on 'optimality principle' which implies that whatever the state or the actions has been taken the remaining states should be taken optimally, so, it is divided the problem to states then find the optimal reward that could be getten, it is related to the definition of both the Q-value function and value function as it divided the dynamic problem recursively to sub tasks or to be accurate the definition of Q-value function coming from this Bellman Equation. So, the bellman equation for our Q-value function in RI[4]:

$$V^{\Pi^*}(s) = \max_a \left\{ R(s,a) + \gamma \Sigma_{s'} P(s'|s,a) \times V^{\Pi^*}(s') \right\}$$

Where:

$V^{\Pi^*}$ = Value function
s' = Next state
$\pi^*$ = Optimal policy
$\gamma$ = Discount factor
P(s'|s, a) = Probability to go to state s' when you take action a in s state
R(s, a) = Reward function of taking action a in state s

## VALUE ITERATION

It is the process of exploring the expected reward function R for each state s using bellman optimization equation until finding a good policy (converge the value function) to follow, it is start with empty value function and then keeps evaluation this function and improves it while exploring and taking experience from the environment, here is the pseudo code to find the policy using value iteration:

Initialize the value V to random values
Repeat
  for all State s∈S
    for all action a∈A
    $Q_\pi(s, a) = E_\Pi[r|s, a] + \gamma \Sigma_{s'} P(s'|s, a) \times V^{\Pi^*}(s')$
  V(s) = $\max_a$Q(s, a) from the for loop
  Until V(s) converge[8]

## POLICY ITERATION

It is similar to Value Iteration while this iteration works on the Q-value functions directly rather than the value function as we only care about the policy, we converge the Q-value directly, it needs less iteration to converge but each iteration more complex than the one used in value iteration. It is useful as sometimes the convergence of the policy happens before the converages of the value function itself. Here, is the pseudo code:

Initialize the policy $\Pi^*$ to random values
Repeat
  $\Pi = \Pi^*$
  $V_\Pi = E_\Pi[r|s, \Pi(a)] + \gamma \sum_{s'} P(s'|s, \Pi(a)) \times V_{\pi^*}(s')$
  $V^{\Pi^*}(s) = \arg\max_a \{E[R(s,a)] + \gamma \sum_{s'} P(s'|s,a) \times V^{\Pi^*}(s')\}$
  Until V(s) converge

## Q-LEARNING

It is a model free learning in which the agent doesn't know anything about the states, rewards, or the environment. So, the agent will get its experiences from interacting with the environment. It is used time difference learning which is approximation for the policy, so it keeps update the Q-value after each iteration using this equation:

$$Q(s,a)=(1-a)Q(s,a)+\alpha Q_{obs}(s,a)$$

Where:

$$Q_{obs}(s,a)=r(s,a)+\gamma \max_{a'}(s',a')$$

So, after that deep network used to approximate this Q value function to be used for the agent to take a good actions[2]. Here, is the pseudo code for Q-learning algorithm that we used in building our agent[2]:

```
Initialize Q(s, a) Randomly, D ={}
Repeat for all episode:
    {  initialize S
       for each step in the current episode
       {
          r = generate Random(0,1)
          if(r>∈)a_t = Random Action()
          else a_t = Q*(s_t, a, θ)
          execute a_t
          store transition (s_t, a_t, r_t, s_t+1) in D
          s_t+1 = s_t
          sample random minibatch of the transition
          set y_j = r_j = γmax_a' Q(s_t+1, a'; θ)
          or y_j = r_j forn terminal state
          define loss as L_θj = (y_j-Q(s_t, a_j; θ))
          perform adam gradient descent on L_θj
    }
}
END
∈:Exploration rate it should keep decreasing.
D:replay memory
```

As you can see the Q learning keeps updating the weights after each iteration until it is coverage and the neural network gives a good approximation for the Q value function so that we can use as a policy.

## EXPLORATION VS EXPLOITATION

Exploration is the process discovering the environment by the agent so it starts by taking somehow random actions to build a knowledge and experience about the environment that will help him to build the policy, exploitation is the process of using this knowledge to take a good actions, so, the agent should start with hight exploration and keep building his knowledge then he should balance between exploration and exploitation to build a good policy, so that, we use monotonic decreasing function to reduce exploration rate.

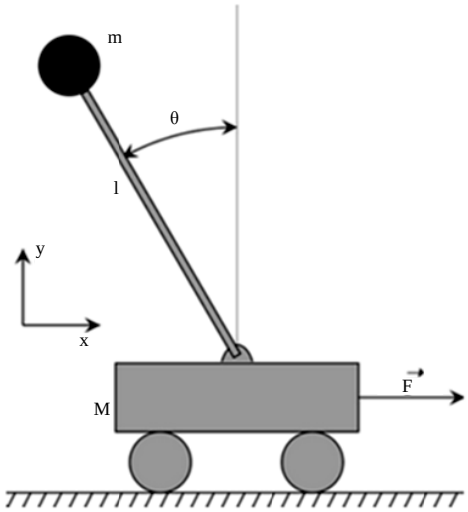


Fig. 2: Cart Pole

## CART POLE AGENT

**About the game:** Before talking about our agent we are going to talk about the game itself, how to play, its rules like when you win, legal moves, etc.

As seen in Fig. 2, there is a cart and in the middle of it there is a pole, the idea is to keep the pole vertically balanced and avoid dropping it down. The player can move the cart either to the right or left and he will loss when $\theta>15$ or $\theta\leq15$ or the cart goes to the left or right most of the screen, so you need to keep moving the cart left and right in the middle of the screen and balance the pole at the same time to win the game.

**Environment:** We used python as programming language, it is a rich, good tool for machine learning and reinforcement learning. I used Jupyter notebook to write the code and run it.

We used keras library to train our agent and build the neural network that will be used to approximate the policy function. We used GYM library, it provides you with the game itself and a lot of API that is built in specially for Reinforcement learning purpose.

We have three layers network to approximate the policy function. I used my machine to for learning and testing the agent.

So, the agent trained for 600 games and he keeps improving his policy, as seen in Table 1 after each 50 games we saves the policy that was built by the agent to compare between them and here are the results.

**Agent:** Our agent use Q-learning algorithm to build its policy, it uses three layers as neural network that used to approximate its policy function, it starts with exploration rate 95% and keep decays while building its policy (using

Table 1: Result of the training

| After x trained games | Mean of 100 test games | SD of 100 test games | No. of completed games | Comment |
|---|---|---|---|---|
| 0 | 8.36 | 0.728 | 0 | Random Agent |
| 50 | 8.45 | 0.739 | 0 | Better but still likes random agent |
| 100 | 35.05 | 13.067 | 0 | It seems the agent starts learning jump from 8-35 |
| 150 | 54.12 | 14.700 | 0 | It is doing better |
| 200 | 24.11 | 3.720 | 0 | Still explores |
| 250 | 152.62 | 7.560 | 0 | Again it is doing better |
| 300 | 234.57 | 15.500 | 0 | As mean is 234 it is above the normal game which is 200 |
| 400 | 217.02 | 13.920 | 0 | It seems the agent still explore |
| 450 | 230.39 | 31.500 | 1 | High standard deviation some games high some games low (exploration vs exploitation) and finish one game |
| 500 | 1246.71 | 772.000 | 5 | Keeping the pole more than one time but high SD (exploration) |
| 600 | 3000 .00 | 0.000 | 100 | It completes all the games |

monotonic decreasing function) but the agent will keep exploration even after building his network with rate 0.001, learning rate was set to 0.0001 and we use Adaptive Moment Estimation (Adam) to optimize the weight for the neural network after each step.

The game will be ended if the agent fails to keep the pole balanced or go out of the screen, left or right or if the agent keeps the pole balanced for >200 time steps, we have hacked the game and changed this, so, he can keep playing util to 3000 timesteps, this is helpful for testing phase as we need to check if our agent works well enough and compare between training sets, here is a photo from the game in the GYM library as it seems different than Fig. 3 but they have the same idea, same rules and same functionality.

## OUR OBSERVATION

Agent training live time is briefly described in Table 1, the random agent can't keep the ball balances even for 10 timesteps and even after 50 games almost the agent has the same abilities but there is a good improvement about 400% after it trained for 100 games and keeps improvement after 150 but there is a decay after 200 games, you can see after 500 games the agent gets a good score about 1246 and finished some games, after 600 it gets its optimal policy and success in all trails, you can check the comments for each trails in the table for more details and Fig. 4 that shows the enhancement the agent got while training.

Watching the agent while training is really interesting, started randomly and fails here and there after that you feel like it really starts to understand the environment and to learn from it. it starts keeping the pole up but still he does a lot of mistakes and keeps balance between exploring the environment and use what he was learnt after 500 games he reaches a good policy he keeps vibration but goes to the left or right most of the screen and this makes him loose after 600 he understood how to win, he should keep the pole balances in the middle of the

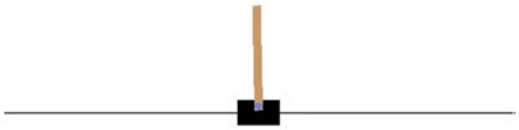

Fig. 3: Cart Pole from GYM library

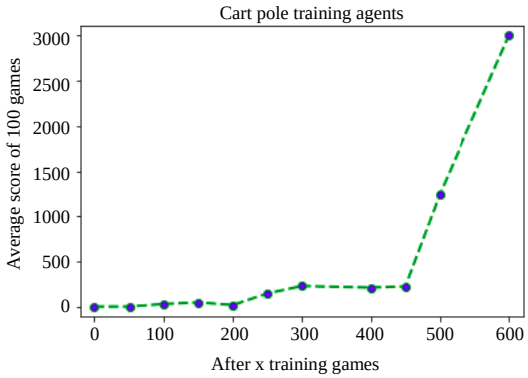


Fig. 4: Training set vs. performance

screen so as you can see there is a clear jump after 450 training games and after 600 games the agent surpasses the human being.

## REFERENCES

01. Samson, L., 2016. Deep reinforcement learning applied to the game bubble shooter. BA. Thesis, University of Amsterdam, Amsterdam, Netherlands.
02. Sutton, R.S. and A.G. Barto, 1998. Reinforcement Learning: An Introduction. 1st Edn., MIT Press, Cambridge, MA.,.
03. Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, 2013. Playing atari with deep reinforcement learning. Comput. Sci., Vol. 2013.
04. ElMaraghy, H.A., 1987. Artificial intelligence and robotic assembly. Eng. Comput., 2: 147-155.

05. Silver, D., A. Huang, C.J. Maddison, A. Guez and L. Sifre *et al*., 2016. Mastering the game of go with deep neural networks and tree search. Nat., 529: 484-489.
06. Saito, S., Y. Wenzhuo and R. Shanmugamani, 2018. Python Reinforcement Learning Projects: Eight Hands-on Projects Exploring Reinforcement Learning Algorithms using Tensor Flow. Packt Publishing Ltd, Birmingham, UK., ISBN: 9781788993227, Pages: 296.
07. LeCun, Y., Y. Bengio and G. Hinton, 2015. Deep learning. Nature, 521: 436-444.
08. Goodfellow, I., Y. Bengio and A. Courville, 2016. Deep Learning. MIT Press, Massachusetts, United States, ISBN: 9780262337373, Pages: 800.
09. Bellman, R., 1957. A markovian decision process. J. Math. Mech., 6: 679-684.
10. Bellman, R.E., 1957. Dynamic Programming, ser. in Rand Corporation Research Study. Princeton University Press, Princeton, New Jersey,.