International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 06 Issue: 04 | April - 2022 Impact Factor: 7.185 ISSN: 2582-3930

SOLVING A TWO DIMENSIONAL ENVIRONMENT WITH REINFORCEMENT LEARNING

Sanjay Dokula[1], Sahithi Marella[2], Ritesh Rokkam[3], G.Surya Bharti[4].
[1,2,3]Student, Department of Computer Science Engineering, Gitam University , Visakhapatnam.
[4]Assistant Professor, Department of Computer Science Engineering, Gitam University , Visakhapatnam.

ABSTRACT
Reinforcement learning (RL) is a branch of machine learning that studies how intelligent agents should operate in a given environment in order to maximise cumulative reward. Along with supervised and unsupervised learning, reinforcement learning is one of three basic machine learning paradigms. and unsupervised learning. In this project, we show how to make a Reinforcement Learning agent learn in any environment that it is put in and demonstrate why Q-learning is not advisable to use in a high dimension environment or an environment with continuous observation space with relatively big intervals. In this process, we will come across OpenAI/Gym and stablebaselines3 in implementing the above concept
To ease usage, we have also developed a Graphical Interface using the PyQt Library, a ported version from the c++ Qt library to python.

1. INTRODUCTION

Reinforcement Learning (RL) is a machine learning technique in which an agent learns the best action to take for a given task by interacting with a dynamic environment that either rewards or punishes the agent's actions. Reinforcement learning is a semi-supervised learning method in which the model's cost/loss value is delivered indirectly through the environment's incentives. Reinforcement learning is more suited to learning dynamic environmental interactions than static patterns between two sets of input and output values. Many reinforcement learning approaches and architectures have been developed throughout the years, with varied degrees of success. The recent success of deep learning algorithms, on the other hand, has reignited interest in reinforcement learning, which is currently being utilised to solve extremely difficult problems that were previously thought to be unsolvable [1]. Artificial agents such as AlphaGo [3] [9] beating world champion Lee Sedol [3] [9] or IBM Watson [5] [14] winning the game of Jeopardy [5] [14] have drew international attention to the emergence of artificial intelligence, which may soon surpass human intelligence [11]. [4]. Reinforcement learning is crucial To create intelligent systems that can learn from their experiences throughout time, a new paradigm is needed. Robotics, healthcare, recommender systems, data centres, smart grids, financial markets, and transportation are all using reinforcement algorithms currently [13].There are two sections that work together; the dynamic environment and the RL agent that plays the environment to learn the optimal policy. Policy here is the function that maps the states and actions to maximise the cumulative reward.

2. LITERATURE SURVEY

Guillaume Lample, et al. presented the first architecture to tackle 3D environments in first-person shooter games, that involve partially observable states. The architecture substantially outperformed the built-in AI agents of the game as well as humans in deathmatch scenarios.
In 2013, **Volodymyr Mnih et al**. published the first deep learning model that effectively learned control policies from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network trained using a Q-learning variation, using raw pixels as input and a value function forecasting future rewards as output.
Swagat Kumar provided the comparisons between Q-learning and DQN in a 2D environment of continuous observation space called a cart pole system. This paper showed the shortcoming of Q-learning in continuous observation space environments and how the DQN overcame this problem.

Łukasz Kaiser et al. has published a paper showing how a model-based reinforcement learning agent can solve the Atari games with fewer interactions than a model-free method. They described Simulated Policy Learning, a complete model-based deep RL algorithm based on video prediction models.

Jason Rennie, in his work, presented a novel way of creating web spiders using reinforcement learning and argues that it is the best way to do it.

D. M. Roijers et al. look at methods for multiple-objective sequential decision-making situations. Despite the fact that there is a growing volume of literature on the issue, little of it specifies when unique methods are required to address multi-objective problems. As a result, we identify three instances in which reducing a multi-objective problem to a single-objective problem is either impossible, infeasible, or undesirable.

When rewards are delayed and sparse, Reinforcement Learning (RL) algorithms might suffer from poor sample efficiency. **Andrew Levy et al**. presented a method for agents to learn temporally extended actions at several levels of abstraction in a sample efficient and automated manner. Our method combines universal value functions and hindsight learning, allowing agents to simultaneously learn policies over many time scales. In a range of discrete and continuous tasks, we show that our strategy dramatically accelerates learning.

Lucian Buşoniu et al. in this study presents a thorough examination of multiagent reinforcement learning (MARL). The formal description of the multiagent learning goal is a key topic in the area. Different perspectives on this subject have resulted in the formulation of numerous goals, two of which stand out: the stability of the agents' learning dynamics and adaptation to the changing behaviour of other agents.

3. METHODOLOGY

3.1 Reinforcement Learning:

The term "reinforcement learning" refers to a framework for learning optimal decision making from rewards or punishment [Kaelbling et al., 1996]. It differs from supervised learning in that the learner is never told the correct action for a particular state, but is simply told how good or bad the selected action was, expressed in the form of a scalar "reward." A task is defined by a set of states, s ∈ S, a set of actions, a ∈ A, a state-action transition function, T : S ×A → S, and a reward function, R : S ×A → < 1, devalues rewards received in the future. Accordingly, when following policy π, we can define the value of each state to be: V π (s) = X∞ t=0 γ t rt, (1) where rt is the reward received t time steps after starting in state s and following policy π. The optimal policy, written π ? , is the one that maximizes the value, V π (s), for all states s.

3.1.1 Q-learning

The creation of an off-policy TD control method known as Q-learning (Watkins, 1989) was one of the early triumphs in reinforcement learning.

$$Q(s_i, A_i) = Q(s_i, A_i) + a\left[R_{t+1} + \gamma \max_a Q(S_{i+1}, a) - Q(S_i, A_i)\right]$$

In this scenario, regardless of the policy used, the learned action-value function, Q, directly approximates q, the ideal action-value function. This greatly simplifies the algorithm's analysis and allows for early convergence proofs. In that it affects which state–action pairings are visited and altered, the policy still has an impact. All that is required for proper convergence is that all pairings be updated continuously.

Algorithm:

Initialise Q(s,a) for all s ∈ S

Loop for each episode:

Initialise S

Loop for each step of episode:

Choose A from S using policy derived from Q

Take action A, observe R, S'

$$Q(s_i, A_i) = Q(s_i, A_i) + a\left[R_{t+1} + \gamma \max_a Q(S_{i+1}, a) - Q(S_i, A_i)\right]$$

S = S'

Until S is terminal

3.1.2 Sarsa

The state–action–reward–state–action (SARSA) method is a machine learning reinforcement learning approach for learning a Markov decision process policy. Rummery and Niranjan offered the

moniker "Modified Connectionist Q-Learning". Rich Sutton submitted the alternate name SARSA,

This name simply reflects the fact that the main function for updating the Q-value is dependent on the agent's current state "S1," the action "A1," the reward "R" the agent receives for taking that action, the state "S2" that the agent enters after that action, and finally the next action "A2" the agent chooses in its new state.

Q(St, At) Q(St, At) + ↲ h Rt+1 + Q(St+1, At+1) Q(St, At) i .

An on-policy learning algorithm is one in which a SARSA agent interacts with the environment and adjusts the policy based on the actions made. An mistake, modified by the learning rate alpha, updates the Q value for a state-action. The Q values represent the potential reward for taking action an in state s in the next time step, as well as the discounted future benefit from the next state-activity observation.

Based on the maximum reward of possible actions, Watkin's Q-learning updates an estimate of the best state-action value function displaystyle Q*Q*. Watkin's Q-learning learns the Q values associated with taking the optimal policy while following an exploration/exploitation strategy, whereas SARSA learns the Q values associated with taking the policy it follows.
Loop for each episode:
Initialize S
Choose A from S using policy derived from Q (e.g., "-greedy)
Loop for each step of episode:
Take action A, observe R, S0 Choose A from S using policy derived from Q (e.g., "-greedy) Q(S, A)= Q(S, A) + alpha [R + gammaQ(S0 , A0) -Q(S, A)]
S =S0 ; A= A0 ;
until S is terminal.

3.2 Environment

A gym will essentially be a class with four functions. The class's initialization function is the first function, which takes no more parameters and initialises the class. It also establishes the starting point for our RL dilemma. The step function takes an action variable and returns a list of four items: the next state, the current state's reward, a boolean indicating whether the current episode of our model is complete, and some additional information about our problem. The other functions are reset, which returns the environment's state and other variables to their initial values, and render, which displays pertinent information about our environment's activity thus far.

3.3 Experimental Setup

We chose a desktop PC with a competent GPU and a respectable frame rate to provide us with an appropriate computing platform. Table 1 lists the characteristics of this machine. The acceleration of the NVIDIA Pascal Architecture is used. This GPU can speed neural network models because to its 768 CUDA cores and 4 GB of GDDR5X RAM memory.
Table 1: Desktop Characteristics

CPU	Intel i5 @ 3.7 GHz
GPU	Nvidia 1050Ti
RAM	16 GB

4. IMPLEMENTATION

First researched on search algorithms and found out that they do not perform like a human.So looked at cnn approach for human like behaviour. To train, it requires a large amount of data. We need an approach that needs to work without any previous data and interact with a dynamic environment.
RL techniques are the best suited for this kind of problem. After testing algorithms like Q-learning, DQN, and some policy gradient methods found out that Q-learning is not suitable for high dimension observation space environment, DQN does not converge always and needs add ons, PPO a proximal policy optimization although takes a bit time but always converges to the optimal policy. The reward system was revised multiple times to adjust the agents behaviour for the environment. First it was same the game score which did not give any viable result, then tried out period rewards for getting a score, this had a bit progress in terms of moving toward the goal but still not acceptable. Finally tested another reward system based on Euclidean distance.
Reward system:
sqrt((snake_head.x – apple.x)/(snake_head.y – apple.y)) whole squared

5. RESULTS

After testing algorithms like Q-learning, DQN, and some policy gradient methods found out that Q-learning is not suitable for high dimension observation space environment, DQN does not converge always and needs add ons, PPO a proximal policy optimization although takes a bit time but always converges to the optimal policy.

The reward system was revised multiple times to adjust the agents behaviour for the environment. First it was same the game score which did not give any viable result, then tried out period rewards for getting a score, this had a bit progress in terms of moving toward the goal but still not acceptable. Finally tested another reward system based on Euclidean distance between agent and the goal and got a very solid result.

Fig. 3: Episode reward mean rollout graph

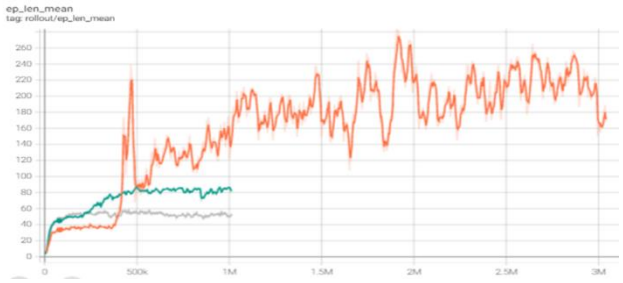

Fig. 4 : Episode length mean

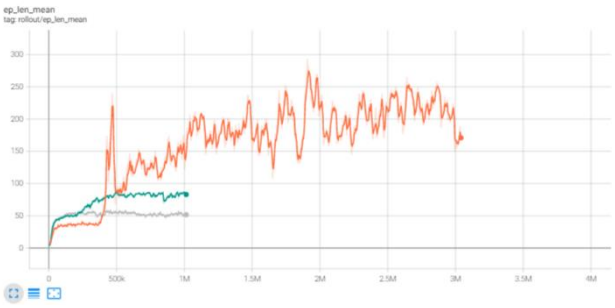

Fig. 5 : Renderings from the environment

6. CONCLUSION

Reinforcement Learning is the newest and most promising for of machine learning method to train the behaviour of control systems like network routing, game AI, human brain replicating, etc. This project can further be pushed to 3D environments. There are a lot of domains where RL can do a much better and efficient job than existing approaches, for example we can simulate a football game to see the many ways the agent tries as it is very fast in computations and we can evaluate a few interesting startegies to use. Another can be in fluid mechanics, we can make a simulation to create a car chasis that the agent can modify. This can lead to very interesting and innovative also naïve strategies and models to increase the aerodynamics or drags or whatever the critera need.

7. REFERENCES

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine, 34(6):26–38, 2017.

[2] E. Bisong. Google colaboratory. In Building Machine Learning and Deep Learning Models on Google Cloud Platform, pages 59–64. Springer, 2019.

[3] S. Borowiec. Alphago seals 4-1 victory over go grandmaster lee sedol. The Guardian, 15, 2016.

[4] J. Fang, H. Su, and Y. Xiao. Will artificial intelligence surpass human intelligence? Available at SSRN 3173876, 2018.

[5] D. A. Ferrucci. Introduction to this is watson. IBM Journal of Research and Development, 56(3.4):1–1, 2012.

[6] Google Colaboratory. Online gpu cloud by google. https:// colab.research.google.com/.

[7] A. Gulli and S. Pal. Deep learning with Keras. Packt Publishing Ltd, 2017.

[8] H. V. Hasselt. Double q-learning. In Advances in neural information processing systems, pages 2613–2621, 2010.

[9] S. D. Holcomb, W. K. Porter, S. V. Ault, G. Mao, and J. Wang. Overview on deepmind and its alphago zero ai. In Proceedings of the 2018 international conference on big data and education, pages 67–71, 2018.

[10] Kaggle. Online gpu cloud with datasets. https://www.kaggle. com/.

[11] P. Kraikivski. Seeding the singularity for ai. arXiv preprint arXiv:1908.01766, 2019.
[12] S. Kumar. Reinforcement learning code for cartpole system. https: //github.com/swagatk/RL-Projects-SK.git, 2020.
[13] Y. Li. Reinforcement learning applications. arXiv preprint arXiv:1908.06973, 2019.