International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 10 (2018) pp. 8372-8376

Automated Virtual Car Simulation using Deep Reinforcement (Q) Learning

Indraneel Brahma
Vellore Institute of Technology - University in Vellore, Tamil Nadu, India.

Sandipan Basak
Vellore Institute of Technology - University in Vellore, Tamil Nadu, India.

Soumallya Boral
Vellore Institute of Technology - University in Vellore, Tamil Nadu, India.

Satyaki Mukherjee
Vellore Institute of Technology - University in Vellore, Tamil Nadu, India.

Under the guidance of
Dr. Swarnalatha P (Associate Professor)
School of Computer Science and Engineering
Vellore Institute of Technology - University in Vellore, Tamil Nadu, India.

Abstract

This paper describes the simulation of an automated virtual car which is developed using a machine learning algorithm called Deep Reinforcement (Q) learning. The basic motive of this paper is to develop a self-driving car where the source and destination would be fixed and based on the data it receives from the environment it continuously trains itself to reduce the mistakes.

Keywords: Deep Reinforcement learning, machine learning, automated

LITERATURE REVIEW

a) Playing Atari with Deep Reinforcement Learning

This paper shows the primary significant learning model to viably learn control approaches particularly from high-dimensional material data using bolster learning. The model is a convolutional neural framework, organized with an assortment of Q-learning, whose data is rough pixels and whose yield is a regard work studying future prizes. We apply our technique to seven Atari 2600 beguilements from the Arcade Learning Environment, with no difference in the building or learning estimation .We apply our method to seven Atari 2600 beguilements from the Arcade Learning Environment, with no change of the building or learning estimation. The model beats every single past philosophy on six of the redirections and beats a human ace on three of them.

b) Human-level control through deep reinforcement learning

In this paper, we use late advances in getting ready significant neural frameworks to develop a novel recreated administrator, named a significant Q-mastermind that can increase successful game plans clearly from high-dimensional unmistakable wellsprings of data using end-to-end fortress learning. We attempted this administrator on the testing zone of awesome Atari 2600 amusements. We show that the significant Q-organize master, tolerating only the pixels and the preoccupation score as inputs, could beat the execution of all past counts and achieve a level equivalent to that of a specialist human-beguilement's analyser over a set of 49 entertainments, using a comparable figuring, arrange outline and hyper parameters. This work traverses the hole between high-dimensional material wellsprings of data and exercises, achieving the first fake pro that is prepared for making sense of how to surpass desires at a different bunch of testing errands.

c) Reinforcement Learning Methods for Continuous-Time Markov Decision Problems

Consequent to investigating semi-Markov Decision Issues and Bellman & optimality condition in that setting, this paper has made a couple of figures like those named above, acclimated to the plan of semi-Markov Decision Problems. This paper shows the estimations like semi-Markov Decision Problems and Bellman & optimality condition in that extraordinary condition, by applying them to the issue of choosing the perfect control for a fundamental queueing system and along these lines close with a talk of conditions under which these figures may be advantageously associated.

d) Reinforcement Learning in Finite MDPs: PAC Analysis

This paper looks at the issue of learning close perfect lead in restricted Markov Decision Processes (MDPs) with a polynomial number of tests. These "PAC-MDP" counts join the definitely comprehended E3and R-MAX estimations and furthermore the later Delayed Q-learning count. The paper gathers the present best in class by presenting limits for the issue in a united speculative structure. A better examination for upper and lower limits is acquainted with yield information into the complexities between the without display Put off Q-learning and the model-based R-MAX.

FRAMEWORK

Markov decision strategies (MDPs) give a logical framework for showing decision making in conditions where comes about are not entirely unpredictable and mostly under the control of a decision maker. MDPs are significant for think a broad assortment of streamlining issues clarified by methods for dynamic programming and bolster learning. MDPs were alluded to in any occasion as in front of calendar as the 1950s (cf. Bellman 1957); a middle variety of research on Markov decision methods happened in light of Ronald A. Howard book dispersed in 1960, Dynamic Programming and Markov Processes. They are used as a piece of a wide domain of disciplines, including mechanical self-rule, customized control, money related perspectives, and amassing. More precisely, a Markov decision process is a discrete time stochastic control process. At each time step, the method is in some state 's', and the boss may pick any movement 'a' that is open in state 's'. The system responds at the next time progress by self-assertively moving into another state s1, and giving the boss a relating repay. The probability that the methodology moves into its new state s1 is influenced by the picked movement. Specifically, it is given by the state change work P a (s , s1). Along these lines, the accompanying state s1 depends on the present state s and the decision maker's action a. In any case, given s and an, it is prohibitively self-ruling of each and every past state and exercises; in distinctive words, the state changes of a MDP satisfies the Markov property.

Markov decision strategies are an extension of Markov chains; the refinement is the extension of exercises (allowing choice) and prizes (giving motivation). Then again, if only a solitary movement exists for each state (e.g. " wait") and all prizes are the same (e.g. "zero"), a Markov decision process reduces to a Markov chain.

METHODOLOGY

For AI to be considered extremely sharp they should surpass desires at a wide variety of assignments that are considered striving for individuals. Until this point, it had quite recently been possible to make individual computations prepared for acing a single specific region. We use a library in python called Kivy on windows to gather a virtual car expo which wanders discretionarily on a dull screen on the PC, use Pytorch and Deep Q-Learning, i.e. significant Fortress learning through different library in python to set up the auto by giving a psyche to the auto which will impact it to take after a road structure from the source to the objective and a short time later and we may complete the whole undertaking on Linux or Ubuntu

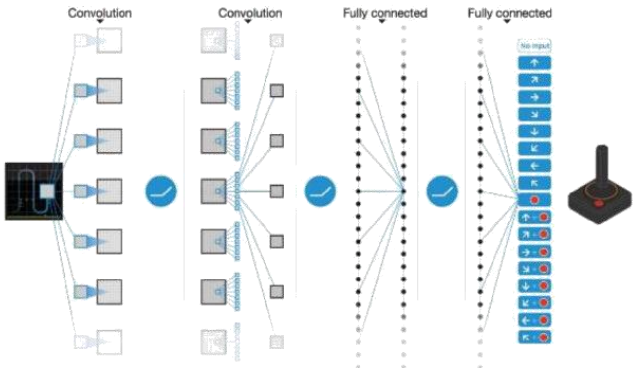

Figure 1: Various Convolutional Layers showing the actual Architecture of the Learning Process

Addition 1: Convolutional Layers

Since our prouct will make sense of how to play videogames, it must have the ability to understand the diversion's screen yield in a way that is in any occasion like how individuals or other clever animals can. Instead of considering each pixel openly, convolutional layers empower us to consider locale of a photo and keep up spatial associations between the things on the screen as we send data up to greater measures of the framework. In this manner, they act so additionally to human open fields. In all actuality there is an accumulation of research exhibiting that convolutional neural network learns depictions that resemble those of the primate visual cortex. Everything considered, they are great for the underlying couple of parts inside our framework.

In Tensor flow, we can use the tf.contrib.layers.convolution2d ability to successfully make a convolutional layer. We make for fill in as takes after: convolution_layer = tf.contrib.layers.convolution2d(inputs,num_outputs, kernel_size,stride,padding)

Here num_outs suggests what number of channels we would apply to the past layer. kernel_size suggests to how immense a window we should need to slide over the past layer. Walk proposes what number of pixels we need to skip as we slide the window over the layer. At long last, cushioning suggests whether we require our window to slide over fundamentally the base layer ("Honest to goodness") or consolidate padding around it ("SAME") sought after to guarantee that the convolutional layer has an Indistinguishable estimations from the past layer. For additional data, see the Tensorflow documentation.

Addition 2: Experience Replay

The second noteworthy development to impact DQNs to work is Experience Replay. The basic idea is that by securing an expert's experiences, and after that subjectively drawing packs of them to set up the framework, we would more have the capacity to healthily make sense of how to perform well in the endeavor. By keeping the experiences, we draw self-assertive, we keep the sort out from simply getting some answers concerning what it is in a split second doing in nature and

empower it to pick up from a more varied display of past experiences. Every last one of these encounters are secured as a tuple of <state, action, reward, next state>. The Experience Replay reinforce stores a settled number of later recollections, and as new ones come in, old ones are expelled. Precisely when the time comes to set us up, essentially draw a uniform group of self-emphatic recollections from the help, and set up our structure with them. For our DQN, we will produce an immediate class that handles securing and recovering memories.

Addition 3: Separate Target Network

The third real expansion to the DQN that makes it one of a kind is the use of a moment arrange amid the preparation methodology. This second system is utilized to produce the objective Q esteems that will be utilized to process the misfortune for each activity amid preparing. For what reason not utilize simply utilize one system for the two estimations? The issue is that at each progression of preparing, the Q-system's esteems move, and in the event that we are utilizing an always moving arrangement of qualities to alter our system esteems, at that point the esteem estimations can without much of a stretch winding wild. The system can move toward becoming destabilized by falling into criticism circles between the objective and evaluated Q-values. With a specific end goal to alleviate that hazard, the objective system's weights are settled, and just intermittently or gradually refreshed to the essential Q-systems esteems. Along these lines preparing can continue in a more steady way.

Going Beyond DQN

With the additions above,, we have all that we need to replicate the DWN of 2014. In any case, the world moves snappy, and different updates well past the DQN configuration portrayed by DeepMind, have contemplated altogether more significant execution and strength. Before setting up your new DQN on your most adored ATARI delight, I would propose taking a gander at the more a la mode increments. I will give a portrayal and some code for two of them: Double DQN, and Dueling DQN. Both are definitely not hard to execute, and by joining the two techniques, we can accomplish better execution with speedier arranging times.

Double DQN

The standard nature behind Double DQN is that the regular DQN as frequently as conceivable overestimates the Q-estimations of the potential moves to make in a given state. While this would be fine if all activities were continually overestimates likewise, there was motivation to trust this wasn't the condition. You can without a considerable amount of a broaden envision that if certain dangerous activities as regularly as conceivable were given higher Q-values than consummate activities, the manager would experience huge inconveniences dependably taking in the perfect strategy. With a specific extreme goal to audit for this, the producers of DDQN paper propose an immediate trap: instead of taking the most outrageous over Q-values when figuring the objective Q respect for our arranging step, we utilize our basic structure to picked a development, and our objective system to make the objective Q-respect for that activity. By decoupling the activity decision from the objective Q-respect age, we can fundamentally diminish the overestimation, and prepare speedier and all the more dependably. The going with is the new DDQN condition for strengthening the objective respect.

$$\text{Q-Target} = r + \gamma Q(s', \text{argmax}(Q(s', a, \Theta), \Theta'))$$

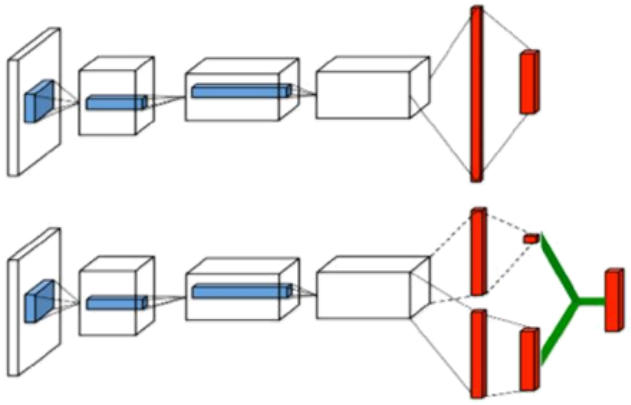

Figure 2: Double Deep Q Learning

With a specific extreme target to clarify the thinking behind the building changes that Dueling DQN makes, we have to first clear up some place in the extent of a couple of extra stronghold learning terms. The Q-values that we have been talking about so far relate to how it is so unbelievable to make a specific move given a specific state. This can be shaped as Q(s,a). This development given state can genuinely be spoiled into two more basic contemplations of basic worth. The first is the respect work V, which says fundamental that it, is so marvellous to be in any given state. The second is the favoured perspective work A(a), which tells how much better making a specific move would be separated from the others. We would then have the ability to consider Q being the mix of V and A. All the more formally:

$$Q(s,a) = V(s) + A(a)$$

The objective of Duelling DQN is to have a structure that openly shapes the immense position and respect breaking points, and joins them once more into a solitary Q-work precisely at the last layer. It might appear, in every way, to be truly purposeless to do this at first look. Why disintegrate a cut off that we will simply store up back? The best way to deal with understanding the favoured point of view is to regard that our fortification learning expert should not think about both respect and perfect position at any given time. For instance: envision sitting outside in a diversion centre seeing the nightfall. It is astonishing, and essentially remunerating to stay there. No move should be made, and it doesn't generally look great to consider the benefit of staying there as being

balanced on anything past the normal state you are in. We can accomplish all the more fit checks of express a propelling power by decoupling it from the need of being joined to particular activities.

PROCEDURE

We have made our Virtual Car utilizing the Kivy Library in Python and the car.kv record is the execution of the car like a demonstrating software.

The map.py is the python file which we run to create the environment of the car including the sand layers and the black road.

This has a lot of modules for the car to maintain, and as required we have implemented more than 60 % of the features: -

1. The environment for the car to self-drive itself, i.e. the road the sand-boundaries that we can add if we want.
2. The 360-degree movement of the car in its environment.
3. The button responses on the map, i.e.

   Clear – To clear the road of any sand-boundaries, that we have drawn on the map using the mouse.

   Save – To save the car along with the map environment that we have drawn on the map including the sand-boundaries.

   Load – To load the AI or the brain of the car inside it so that it follows a path from the source (LEFT-TOP Corner of the map) to the destination (RIGHT-BOTTOM corner of the map).
4. How slow will the car move if it collides with the sand-boundaries and how random will be the movement of the car in general.

We have made our Virtual Car using the Kivy Library in Python and the car.kv file is the implementation of the car similar to a modelling software.

The final implementation of our project is that the car (made out of the Library, Kivy in Python) will follow the sand roads that has been drawn on the map by the mouse pointer. We will first load the AI inside the brain if the car and then draw the sand roads inside the window. The car will first move in random ways to go from the source to the destination, then gradually the car will learn to follow the road. What actually happens is that the car keeps on getting reward after every round trip starting from the top left corner to the bottom right corner and then back and depending on this reward the car suits itself and thus tries to follow a path which will get a better reward. The reward system depends on 2 things. First, is the time taken for the complete trip and the second is that how many times the car is crossing a sand road boundary.

The AI.py is the python file which is attached along with the map.py file to create the main brain of the car, i.e. to teach the car how to move about in its environment. The file has mainly 3 modules:

1. Network Class – This creates the AI network of the Convolutional Neural Network that we are creating using the reinforcement learning.
2. Replay Memory Class – This module helps to push data inside every layer of the CNN that we have created inside the Network Module.
3. The main module or the DQN Class – Here we use a lot of sub-modules:

   Learn Module – It learns from its mistakes and from the negative rewards that it gets every time it does a mistake (e.g., crossing the sand road boundary).

   Update module – It updates itself with the mistakes and the lesson learnt from the mistakes and so, from the next trip onwards it tries to take some road which has lesser negative reward than the one before.

   Score Module – This module is used to calculate the score after every round trip and the score keeps on changing and mostly increases as the car gradually learns the proper roads and thus the rewards become positive.

RESULTS

The result of this algorithm gives us a reward-time graph as shown in Fig 5, showing the reward that is varying with respect to time. The reward usually tars with a negative outcome and then finally it gets a positive value as soon as the car starts following the designated road. The road is drawn randomly on the screen by the user and maintaining that road alignment the car gets trained. The car at the beginning starts to get off the track but after some time, the car starts to follow the designated road as has been shown in Fig 3 and Fig 4, and the reward becomes positive with time.

Figure 3: The car is at the top-left corner of the road (source) and is inside the designated road

Figure 4: The car is at the bottom-right corner of the road (destination) and is inside the designated road

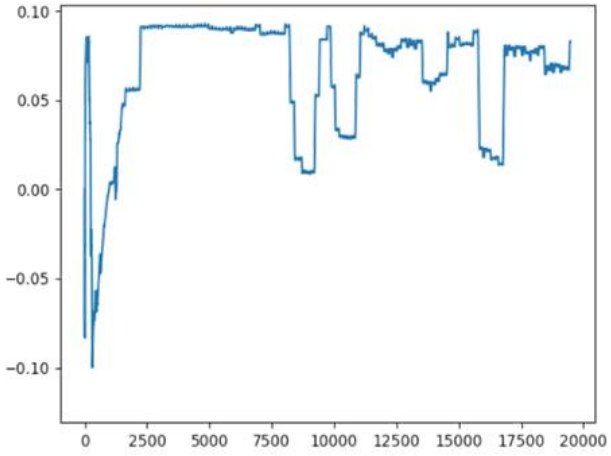

Figure 5: The reward-time graph (reward – y axis and time – x axis), showing the reward gradually going towards a positive value staring from a negative value after the road is drawn on the screen

CONCLUSION

This is a basic simulation of an automated car implemented using Deep Reinforcement Learning which would automatically train itself based on the various data that it gathers from the environment and the obstacles drawn during the training period. This basic prototype can be implemented in a real- world car which can programmed using Arduino or Raspberry Pi hardware devices.

Future Aspects

What we can do to improve this algorithm and implement Google Maps in our simulation. Instead of creating the obstacles by ourselves we would import the google map API where obstacles would present by default and would consider the gradient factor too. The car would then have to follow the actual road as shown by some specific colored roads on the Google map.

We can further work on the velocity of the car. We need to slow down the car when it is at a close proximity of an obstacle so that the whole model is more realistic and the then the final work left to be done would be implementing it on an actual car in the real world or at-least some prototype of a car built with Arduino or Raspberry-pi. We can have similar sensors that we have implemented on the virtual car with the help of various hardware components.

REFERENCES

[1] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In Proceedings of the 12th International Conference on Machine Learning (ICML 1995), pages 30–37. Morgan Kaufmann, 1995.

[2] Marc Bellemare, Joel Veness, and Michael Bowling. Sketch- ased linear value function approximation. In Advances in Neural Information Processing Systems 25, pages 2222–2230,2012.

[3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents.Journal of Artificial Intelligence Research, 47:253–279, 2013.

[4] Marc G Bellemare, Joel Veness, and Michael Bowling. Investigating contingency awareness using atari 2600 games. In AAAI, 2012.

[5] Marc G. Bellemare, Joel Veness, and Michael Bowling. Bayesian learning of recursively factored environments. In Proceedings of the Thirtieth International Conference on Machine Learning (ICML 2013), pages 1211–1219, 2013.