Avoiding obstacles in a road with a maze structure using reinforcement learning methods

Mohamed Saber Rais[a], Rachid Boudour[a] and Khouloud Zouaidia[a]

[a] *University of Badji Mokhtar Annaba, Algeria*

Abstract

Decision making has always been a challenge in all fields and many methods and approaches were applied to solve different kinds of problems in different situations mainly to reach a high level of autonomy in those tasks. In this paper we addressed the autonomous driving problem in a simple way by defining our own environment which represents a road similar to a maze with static obstacles or they can be seen in another way as cars with stable velocities, furthermore we trained each agent in this environment to be able to learn a policy that can let them map between the input data such as the environment state and the actions taken while achieving the best rewards. Using both rule based restrictions and different reinforcement and deep reinforcement learning algorithms we were able to achieve satisfying results in this matter using all five of our agents: the q network agent, the SARSA agent, the deep q network agent, the double deep q network agent and the deep SARSA agent, and we were able to show that the proposed algorithm "deep SARSA with rule based restrictions" can achieve similar or even better results than the q network related methods that are mainly used in autonomy problems.

Keywords 1

Decision making, Reinforcement learning, Deep learning.

1. Introduction

Controlling an agent to do specific tasks holds many challenges and was a field of research for many years.

At first, researchers addressed simpler goals that required just good and clear rules to follow in different situations, this represents the rule based category of the methods that has been used in the last years. But this approach reached its limits after turning our sights to harder tasks in many fields. This is when the notion of learning came to light and we started training those agents to be able to achieve better results in more complicated environments and restrictions.

As for the decision making field, approaches such as deep learning and reinforcement learning were used mainly to be able to reach a human like performance under the "trial and error" method. The last mentioned method let the agent interacts with its surrounding and learn with each attempt what can be best to do in a specific situation till it can find a generalization to what can be done in most situations it will be facing.

This is pretty much what a human does when he tries to achieve a good result in an unknown mission. For example what are addressing now "driving a car", in general with every ride the human takes he learns to adapt more to control the vehicle and can react better to different situations. Same as in this paper with using different techniques of reinforcement and deep Reinforcement learning such as Q network, SARSA, deep Q learning, double deep Q learning and Deep SARSA, each agent learned the

Third conference on informatics and applied mathematics, 21–22 October 2020,Guelma, ALGERIA
EMAIL: Mohamed-saber.rais@annaba-univ.org (A. 1); racboudour@yahoo.fr(A. 2);khouloud.zouaidia@annaba-univ.org (A. 3)
ORCID: https://orcid.org/0000-0002-7706-207X (A. 1); https://orcid.org/0000-0003-2702-8022 (A. 3)
© 2020 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-2748/IAM2020_paper_57.pdf

basics of turning left and right when an obstacle is in front to avoid an accident and reach the destination.
The upcoming parts will represent our work with all the steps taken. Section 2 informs about some prior works related to our subject, section 3 is a background about some concepts we used in our article, section 4 presents a definition to the problem encountered, section 5 contains the results that we reached for every agent, section 6 involves a comparison between the agents used, section 7 holds our final results and finally a conclusion about this subject.

2. Prior Works

The task of decision making was addressed before in many fields and in a lot of dif-ferent ways. We can mention:
In 2013, Mnih V. et al combined deep learning with reinforcement learning to learn a policy that can let the agent play the "Atari game" with human level performance [1].
Another example is the game of "Go" where the authors in 2017 were able to compete against high level players and a variety of computer opponents in this game-board using a reinforcement learning agent [2].
Focusing now on works related to autonomous vehicles:
In 2018, Carl-Johan H. et al used the double deep Q network technique as the base for their agent to reach the wanted level of automation in controlling the car's speed alongside changing the lane in a highway scenario [8].
In 2019, Junjie W. et al combined a high level lateral decision making with low level rule based trajectory modification in a method based on deep Q network to solve the lane changing problem for the autonomous vehicles [9].
In 2019, Laura García C. et al proposed an approach based on using the Q learning algorithm to train their agent in 2 scenarios: the first one was navigating a roundabout with traffic and the second one without traffic. The agent was able to learn smooth and efficient driving to perform maneuvers within roundabouts [10].

3. Background

3.1. Reinforcement learning

is one of the three main paradigms of machine learning alongside supervised learning and unsupervised learning. The goal of reinforcement learning is to learn a policy which maps from the input state to the actions taken by getting the best reward that can be achieved. To obtain this policy reinforcement learning uses a method that maximizes a total expected return (called G) which is a cumulative sum of immediate rewards received over the long run with each action the agent takes. It is defined as follow:

$$G_t = \sum_{k=0}^{\infty} \gamma^k \, r_{t+k}$$

In this formula:
- k represents time-steps.
- r_t represents a reward received at time-step t after applying an action.
- γ represents the discount factor which varies from 0 to 1 and used to quantify the importance for future rewards.

3.2. Deep Reinforcement learning

It uses the principles of both deep learning and reinforcement learning to reach solutions to many problems in different fields but mainly in the autonomy field [1].

3.3. Markov Decision Process

It is a discrete time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker [3].

3.4. Q learning technique

Agents based on this technique use a predefined table which will hold the q values for each (state, action) pair. The q table is updated after each action the agent takes. When taking the action the agent always picks the one with the highest q value in the table for each state [4].

3.5. Deep Q learning technique

This technique has the same concept of the previous one when it comes to taking actions. But instead of a Q table to save the (state, action) pair, a deep learning model is used to save episodes and let the agent learn which action is best to take in a specific situation after checking the saved episodes [8].

3.6. Double deep Q learning technique

In Double Deep Q Learning the agent uses two neural networks to learn and predict what action to take at every step [6].

3.7. SARSA technique

Same as the q learning technique mentioned before agents based on SARSA use a predefined table that will hold the q values for each (state, action) pair and update this table after each action the agent applied.
But when taking every action the agent uses a ε-greedy policy. This means that it can take either a random action that can possibly give some new better results or it can pick the action with maximum q value [4].

3.8. Deep SARSA

Same as the deep q network, agents based on deep SARSA use a deep learning model to save each episode the agent pass by instead of a Q table and when it comes to taking the actions at each state it uses the same concept of the SARSA technique [7].

4. Problem definition

We decided to use a maze representation for our road to let the agents learn the basic actions that can be taken. Thus the problem can be defined as a Markov Decision Process (MDP) where all states are known to the agent.

The agents receive at each discrete time step "t" the state "s" of the road and then select an action to take "a" that leads to the next state "s+1" and saves the reward "r" for taking this action, this continues till an agent reaches a final state, either reaching its goal or losing.

4.1. State space definition

At each time step the agent receives the state of the environment, which in our case is represented with a 2D vector with 3 rows (the figure below shows an example of our environment). It contains the different obstacles and the current placement of our agent after taking any possible action. The obstacles cells were represented with 0 and the free spaces where the agent can travels with 1, our agent was represented with a 0.5 grey color degree to differentiate it from other occupied cells.

Figure 1: Example of our road representation

4.2. Action space definition

Action space is a 1D vector containing 3 actions: forward, left and right, so the agent can move forward or turn left or right at every obstacle. This basically simulates a highway situation where the agent should not move backwards.

4.3. Rules and restriction

For a faster and smoother learning for each agent, we defined some rules in this environment that restricts taking some meaningless actions in some situations (getting off the road) and let the agent focus only on avoiding the obstacles.

Algorithm 1: Rules Function

```
Actions = [0, 1, 2] # actions vector [forward right left]

if row is left line of road:
    actions.remove(2)

elif row is right line of the road:
    actions.remove(1)

if reached the end of the road:
    actions.remove(0)
```

Example: if our agent is localized at the row with index 0 the action turn left will be re-moved from the actions space so we avoid letting the agent go off road. This will let us skip some episodes where

our agent will try to learn actions that are not related to avoiding obstacles and this will lead to a faster learning.

4.4. Rewards

We defined a simple reward function based on 5 situations the agent can be in after taking an action. These situations are: reached the goal, agent is blocked, agent in a visited location, invalid action taken and valid action taken.

Algorithm 2: Reward Function

```
if agent reached its destination :
        return 1.0
if mode == 'blocked':    # no more actions can be taken by the agent
         return self.min_reward – 1

if agent in self.visited : # agent keep turning left and right(no advance)
        return -0.25

if mode == 'invalid':      # agent picked an action that  leads to an accident
        return -0.75

   if mode == 'valid':     # valid action
         return -0.04
```

In our reward function penalizing the agent with "min_reward – 1" when it is in a blocked situation will end directly the episode and starts a new one.
We set a penalization to our agent even when it takes a valid action that leads it to follow the longest road to reach its destination to let the agent knows that it must reach its destination with the least actions possible. In other words, it should try to pick the smallest series of actions that will lead it to reaching its destination while avoiding actions that will lead to consuming more time to finish the episode.

5. Our reinforcement learning and deep R learning agents

5.1. Reinforcement learning agents

5.1.1. Q learning agent

In our experiments we let the agent learn for a total number of 500 episodes for 10 rounds to test the consistency of our results and this led clearly to updating our q table completely with the best actions that can be taken in each state of the environment. We achieved an average reward equal to: -1.1896200000000205 while reaching the best reward possible a lot earlier than expected at around episode number 40 in each round.

Figure 2: QN agent's Rewards Plot

5.1.2. SARSA agent

Same as the q learning agent e let this agent learn for a total number of 500 episodes for 10 rounds too. This led to an average reward equal to: -1.13894000000002 while it reached the best reward possible around the episode 50 each time.

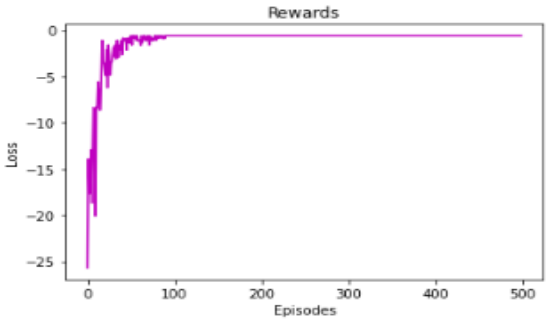

Figure 3: SARSA agent's Rewards Plot

5.2. Deep reinforcement learning agents

We decided to use a simple deep learning model shared between all algorithms mentioned below to see the difference between the performances of these algorithms alone without the interference of the model's architecture.
The model consisted of an input layer that takes the shape of our road as an input and uses the "RELU" activation function with 512 nodes. We used 2 hidden layers for our model each one of them has in order 256 and 64 nodes. The output layer takes the actions vector as an output [5].

5.2.1. Deep Q learning agent

The agent was trained for a total of 500 episodes for 10 times. It was able to achieve the max reward that can be reached in an average number of 250 episodes. The time taken for each round of training was around 9.20 minutes and the agent's loss improved by time. Same as the reward that kept getting bigger till around episode 265 where it got nearly stable with achieving the best reward possible in every run.

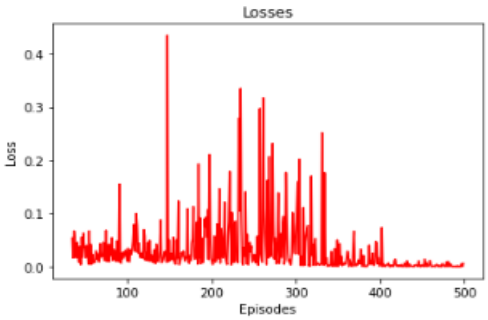

Figure 3: DQN agent's Losses Plot

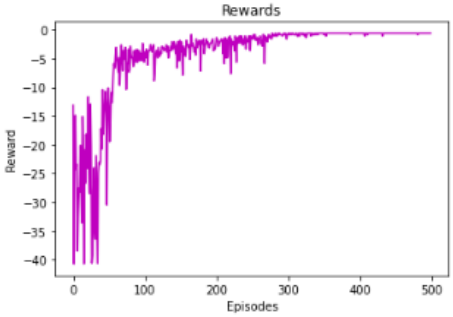

Figure 4: DQN agent's Rewards Plot

5.2.2. Double deep Q learning agent

Same number of episodes and rounds were applied to this agent. It was able to reach the max reward for the first time at the episode 185, with an average of 8.46 minutes for each round. The total reward started getting stable when reaching episode 300 with a really good loss values around that period.

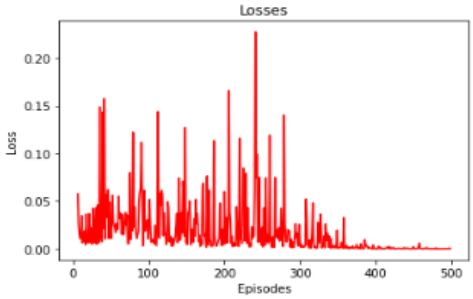

Figure 5: DDQN agent's Losses Plot

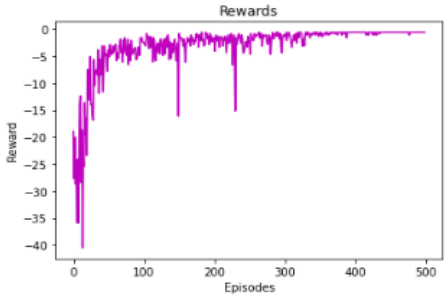

Figure 6: DDQN agent's Rewards Plot

5.2.3. Deep SARSA agent

Lastly we applied the same training amount to the deep SARSA agent that was also able to reach the best achievable reward and this was in general reached at episode 170. The time this agent took for each round of training was close to the previous ones and it was around 7.92 minutes. The loss values got better and better with every episode passing and we were able to reach a nearly stable reward with the best value each time when we got passed episode 230.

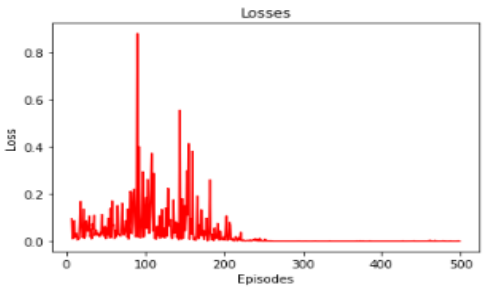

Figure 5: DS agent's Losses Plot

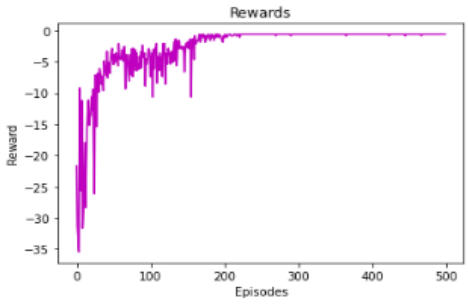

Figure 6: DS agent's Rewards Plot

6. Comparison between the different agents

6.1. Comparison between Reinforcement learning agents

The comparison between the q network agent and the SARSA agent was based on 3 qualities: the time consumed for the training in minutes (TT), average reward (Avg-R) and the last one is the episodes needed to get the best reward (E-BR).

- For the TT quality: whenever the time consumed for the training is smaller it means the agent performed better.
- For the Avg-R: we got it by summing all the reward of each episode the agent finished then dividing by the number of episodes. The agent that will reach the biggest average reward will be the better one.
- For the E-BR: with this metric we want to know which agent is the faster to reach the best reward that can be achieved.

Results are in the table below:

Table 1
Table of comparison between QN and SARSA

Agent	TT	Avg-R	E-BR
Q network agent	2	-1.1896200000000205	**40**
SARSA agent	**1.5**	**-1.13894000000002**	50

While the q network agent was able to reach the best reward faster than the SARSA agent, the later one was able to achieve a better average reward and got to know the best action in each situation a bit faster than the q network agent while at same time taking a significantly less period of time to complete the training.

6.2. Comparison between deep Reinforcement learning agents

The comparison between the deep q network agent, the double deep q network agent and the deep SARSA agent was based on 5 criteria: the average reward achieved (Avg-R), the time consumed needed for training in minutes (TT), the episodes needed to get the best reward (E-BR), number of episodes to reach a stable reward in each episode (SR), loss bound (LB).
The two metrics added here are:

- SR: this metric show when no more improvements can be achieved because we already reached the best possible outcome.
- LB: A value that shows the stability of our training process.

Table 2
Table of comparison between DQN, DDQN and deep SARSA

Agent	Avg-R	TT	E-BR	SR	LB
DQN agent	-4.318239..945	9.20	241-278	280-300	0..-0.4
DDQN agent	-3.121619..947	8.46	182-198	300-350	**0..-0.2**
DS agent	**- 2.752919..935**	**7.92**	**166-196**	**220-270**	0..-0.8

Several information became clear after drawing the table of comparison.
First, results achieved by the 3 agents are all close to each other but the deep SARSA agent was able to get past the other 2 clearly in 4 out of 5 of our defined metrics the Avg-R, TT, E-BR and TT at the cost of a bigger loss interval.
Second, the deep Q network was able to reach a stable reward values every round of training in a small interval of numbers of episodes which means the training for this agent was more stable than the other two.
Lastly the DDQN agent was able to reach the maximum reward for the first time a lot faster than the DQN agent and nearly in the same number of episodes as the SARSA agent, and it has the best loss values of all three agents.

6.3. Comparison between reinforcement learning and deep reinforcement learning agents

In our problem the reinforcement learning agents specifically (q network and SARSA agents) were able to achieve better results mid-training than the 3 deep reinforcement learning agents, due to the fact that our environment is static and doesn't hold that much number of states so a tabular agent can solve this problem faster. But after training our deep learning models they were able to achieve perfect scores each time in our predefined environments and they were able to adapt without any problems to the changes we made later on in our roads.

7. Final results

From the comparisons above we can say that if the goal was just a simple task with limited states there is no need to reach using deep learning models and advanced techniques. But because the final goal is to use each agent in more complicated environments the deep reinforcement learning agents should be a better option. Especially the proposed algorithm "rules based deep SARSA agent" and the double deep q network agent. The last one had the best loss value of all agents but the deep SARSA agent was better in all the rest aspects.

8. Conclusion

This paper demonstrates the power of the reinforcement learning and deep reinforcement learning along-side simple rule based restriction in learning the best policy to navigate a road with a maze structure. We used 5 different reinforcement and deep reinforcement learning algorithms such as the q network, the SARSA algorithm, the deep q network and the double deep q network and lastly the deep SARSA algorithm.

All of our agents which used the previous mentioned algorithms were able to learn a policy that let them achieve the max reward and navigate without any problem in our road. The deep reinforcement learning algorithms also kept a great overall loss values in the middle of the training. Our proposed algorithm "rules based deep SARSA algorithm" was able to achieve great results compared to the more used techniques such as deep q network and double deep q network.

In the future, this work can be extended in several ways such as adding more algorithms for comparison and using a real simulator without changing the basics of these implementations to see how effective these algorithms can be in controlling a real car in a highway situation. This is what is planned for now alongsides some other few ideas of combining other algorithms that are not related to reinforcement learning with the previous mentioned ones.

9. References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller: "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013). arXiv:1312.5602

[2] D. Silver, J. Schrittwieser, K. Simonyan, L. Antonoglou, A. Huang, A. Guez, T. Hubert, T. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van den Driessche, T. Graepel, D. Hassabis : "Mastering the game of Go without human knowledge."Nature 550.7676, pp.354-359(2017). https://doi.org/10.1038/nature24270

[3] https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da, last-accessed 14/05/2020.

[4] S. Ravichandiran : "Hands-On Reinforcement Learning with Python Master reinforcement and deep reinforcement learning using OpenAI Gym and TensorFlow", pp. 91-111. Packt Publishing (2018).

[5] https://pylessons.com/CartPole-reinforcement-learning/, last-accessed 16/05/2020.

[6] Homepage,https://mc.ai/introduction-to-double-deep-q-learning-ddqn/,last-accessed 25/05/2020.

[7] M. Andrecut, M.K. Aliy : "Deep-SARSA: a reinforcement learning algorithm for autonomous navigation", World Scientific Publishing Company, International Journal of Modern Physics C, Vol. 12,No.10,pp:1513-1523(2001).
https://www.worldscientific.com/doi/abs/10.1142/S0129183101002851

[8] W. Junjie, Z. Qichao, Z. Dongbin, C. Yaran : "Lane Change Decision-making through Deep Reinforcement Learning with Rule-based Constraints", International Joint Conference on Neural Networks (2019). arXiv:1904.00231

[9] H. Carl-Johan, W. Krister, L. Leo : "Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning", IEEE International Conference on Intelligent Transportation Systems, pp: 2148-2155 (2018). 10.1109/ITSC.2018.8569568

[10] C. Laura García, P. Enrique, A. Javier Fernandez, A. Nourdine , : "Autonomous Driving in Roundabout Maneuvers using Reinforcement Learning with Q-Learning", Electronics journal (2019). 10.3390/electronics8121536