EcoMotion

Radha Krishna Garg [21BCE7371]
Md. Anas Jamal [21BCE7711]

Abstract

The utilization of artificial intelligence in self-driving systems holds significant prominence. This study integrates the Double-DQN algorithm into our route planning framework, incorporating a novel method for the learning agent to interface with Google Maps. The outcomes reveal a 10% reduction in energy consumption, albeit with a duration twice as long as Google Maps' suggested route. Enhanced results could be achieved through the implementation of intricate neural network architectures, finer navigation resolutions, and unrestricted access to the Google Maps database.

Introduction

Background knowledge

Reinforcement learning (RL) operates as a planning algorithm within the framework of a Markov decision process (MDP). It can be envisioned as an emulation of human decision-making. In this context, a person acts as an agent and confronts situations, such as perusing a restaurant menu. The establishment itself is the environment, while the agent's present circumstances and emotions represent the state. Within a restaurant, decisions are made, leading to actions like placing an order – a term referred to as an action. Interestingly, even if a particular meal has been tried multiple times, there exists a probability that it might turn out exceedingly spicier on a given occasion. This concept is termed state transition probability, denoting that the outcome of the same action in the same state can differ, resulting in distinct flavors (normal or spicy).

Post-meal, a commentary is provided based on the culinary experience, encompassing aspects like taste and the overall dining ambiance. This evaluation corresponds to the reward, which can be positive, signifying a delightful dining experience. Subsequently, this gastronomic encounter influences future decisions when selecting among various restaurants or menu options. Gradually, these patterns of decision-making evolve, shaping what is known as a policy.

This entire process can be encapsulated through Figure 1 and summarized succinctly: within an environment, the agent confronts a state and, guided by a policy, takes an action. The consequences of this action lead to a new state and yield a reward. Another action ensues, followed by another reward and the cycle continues, unfolding through a sequence of state-action-reward iterations. Over time, the agent learns to establish and adapt the policy through this iterative process.

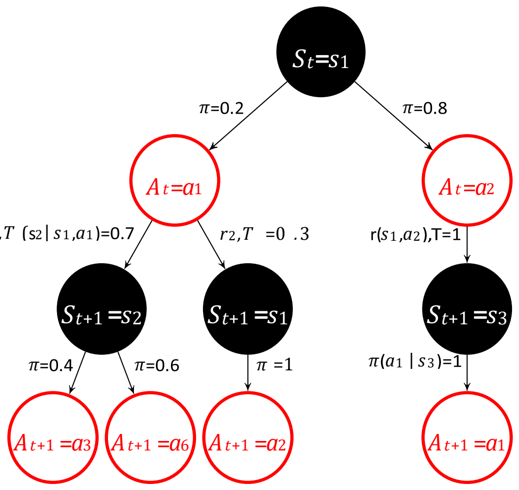


Figure 1: The procedure of iteration between state, action and reward. In state S_t, according to the policy π, we have 0.2 chance to choose action a_1 and 0.8 chance to choose action a_2. The reward r_1 and r_2 can be different.

Some important term and notation we will use in figure 1 and the following paragraph are specified here. • State set S = { s_1, s_2, s_3, ... }

- Action set A = { a_1, a_2, a_3, ... }
- State transition probability function
- T = T(s | s,a) = P[$S_{t+1}=s$ | $S_t = s$, $A_t = a$]. This means the probability of transition from state s to s when taking action a.
- Reward function r = r(s,a) = E[R_{t+1}| $S_t = s$, $A_t = a$]. This means the expected value of reward given s and a.

- Policy $\pi = \pi(a \mid s) = P[A_t = a \mid S_t = s]$. This means the probability of choosing a given the s.
- Discount factor $\gamma \in [0,1]$

Learning Process

Learning from feedback is a fundamental aspect of human decision-making. The objective is to optimize the outcomes towards achieving the ultimate goal. Consider a scenario where a five-star restaurant is situated atop a mountain, and the journey involves potential threats like adversaries and wild creatures. Our aim is to learn a strategy that not only avoids these obstacles but also ensures reaching the mountaintop, as opposed to settling for a two-star restaurant midway.

The learning process of our agent hinges on rewards assigned to each action. We establish an action-value function termed q(s,a), where s represents the state of the agent and a denotes the action taken. If the agent faces four choices of action in a given state s, we compute four q-values. Initially, these q-values are set to 0 and are updated based on encountered rewards. This updating process also considers future rewards, factoring in a discount factor γ that accounts for the significance of future outcomes. A higher value of γ, closer to 1, indicates the comparable importance of future rewards. The combination of current and future rewards can be represented by the equation:

qπ(s, a) =

r(s, a) + γ P s 0∈S T(s 0 |s, a) P a 0∈A π(a 0 |s 0)qπ(s 0 , a 0)

Here, s' signifies the subsequent state relative to the current state s. Since the specific action a' isn't predetermined before the agent enters state s', the expectation of qπ(s', a') is considered through summation and the policy π. Similarly, uncertainty regarding the next state s' after taking action a at state s is addressed using the transition probability T(s' | s, a).

Following the agent's exploration of various states and actions, an instructional map can be constructed to exhibit the desirability of a particular action within a given state. Ultimately, the agent can opt for the action with the highest value in each state, which is likely to lead to the optimal outcome – referred to as the greedy policy.

The updating of q(s, a) can be achieved through off-policy or on-policy methods, differing in how q(s, a) is updated. In the off-policy approach, q(s, a) is updated using maxq(s', a') and r(s, a), while the on-policy approach utilizes q(s', a') and r(s, a).

The value function vπ(s) assesses a specific state by aggregating all q-values associated with that state, formulated as:

vπ(s) = P a∈Aπ(a|s)qπ(s, a)

The same principle applies for this equation as well.

Paper Survey

Deep Reinforcement Learning with Double Q-Learning

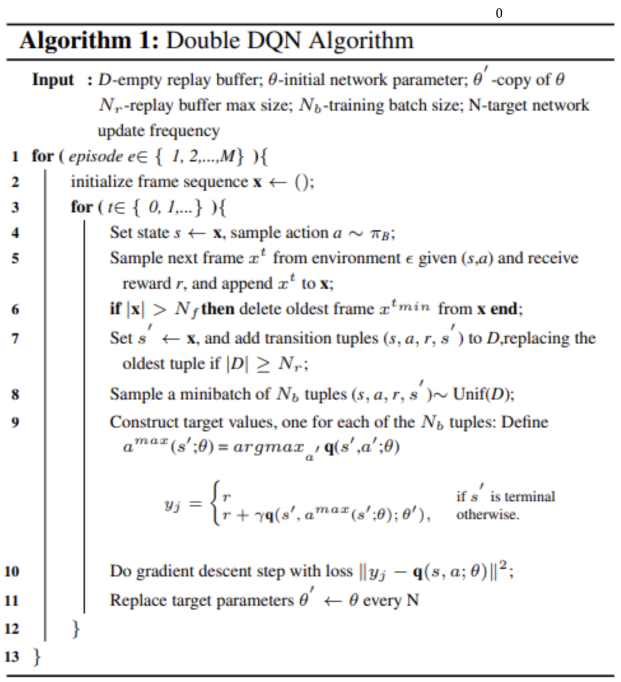
In this study, the utilization of a neural network in conjunction with reinforcement learning is introduced, referred to as the Double Deep Q-Network (Double DQN) method, as proposed by Van Hasselt, Guez, and Silver in 2016. The primary role of the neural network is to facilitate the mapping of the state, denoted as "s," to the corresponding action-values, "q(s, ·)." Notably, the input, "s," can encompass an n-dimensional vector, such as an image, while the output is represented as an m-dimensional vector, where each component signifies a q-value associated with a distinct action. In essence, the neural network functions as a mapping from the n-dimensional real space (IRn) to the m-dimensional real space (IRm). The parameter governing the neural network is denoted as "θ."

A specific objective or target is established to evaluate the q-values computed by the parameter "θ." This target is generated using a separate neural network termed the target network, governed by a distinct parameter denoted as "θ0." The target network shares an identical architectural structure with the primary neural network governed by "θ." The parameter "θ0" is initially initialized to mirror the values of "θ." During the training process within a given episode, wherein the agent navigates from the starting point to the endpoint, the values of "θ" are periodically copied to "θ0" at intervals of N steps. In simpler terms, the parameter "θ0" remains static and is not updated during these N steps. The computation of the target, denoted as "Yt," adheres to the equation:

Yt ≡ rt+1 + γq(St+1, argmax a q(St+1, a; θt), θ0 t).

The process involves inputting the state "St+1" into both the neural network governed by "θ" and the target network governed by "θ0." The highest value in the output vector obtained from the "θ" network, identified as argmax a = q(St+1, a; θt), corresponds to a specific action, "a." Concurrently, the same state "St+1" is inputted into the target network "θ0," yielding another output vector. The target "Yt" is computed by combining the immediate reward "rt+1" with the q-value derived from the target network, q(St+1, a; θ0t).

The learning algorithm outlined in Zyiu Wang's 2016 paper is presented in Algorithm 1.Algorithm 1: Double DQN Algorithm



0

Algorithm 1: Double DQN Algorithm

```
Input : $D$-empty replay buffer; $\theta$-initial network parameter; $\theta'$-copy of $\theta$
        $N_r$-replay buffer max size; $N_b$-training batch size; N-target network
        update frequency
for ( episode $e \in \{1, 2, ..., M\}$ ){
    initialize frame sequence $\mathbf{x} \leftarrow ()$;
    for ( $t \in \{0, 1, ...\}$ ){
        Set state $s \leftarrow \mathbf{x}$, sample action $a \sim \pi_B$;
        Sample next frame $x^t$ from environment $\epsilon$ given ($s$,$a$) and receive
          reward $r$, and append $x^t$ to $\mathbf{x}$;
        if $|\mathbf{x}| > N_f$ then delete oldest frame $x^{t_{min}}$ from $\mathbf{x}$ end;
        Set $s' \leftarrow \mathbf{x}$, and add transition tuples $(s, a, r, s')$ to $D$,replacing the
          oldest tuple if $|D| \geq N_r$;
        Sample a minibatch of $N_b$ tuples $(s, a, r, s') \sim$ Unif($D$);
        Construct target values, one for each of the $N_b$ tuples: Define
          $a^{max}(s';\theta) = argmax_{a'} \mathbf{q}(s',a';\theta)$
          $y_j = \begin{cases} r & \text{if } s' \text{ is terminal} \\ r + \gamma \mathbf{q}(s', a^{max}(s';\theta); \theta'), & \text{otherwise.} \end{cases}$
        Do gradient descent step with loss $\|y_j - \mathbf{q}(s, a; \theta)\|^2$;
        Replace target parameters $\theta' \leftarrow \theta$ every N
    }
}
```

Experience replay is a biological inspired technique to get rid of correlations in the data sequence. We choose data which stored in the replay buffer uniformly at random to compute the loss and updtate the weights during learning.

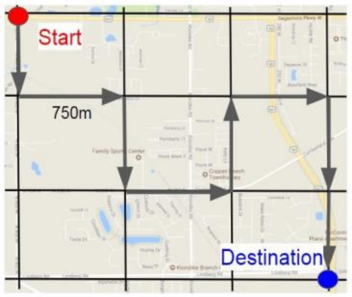
Experiments

The Learning Agent

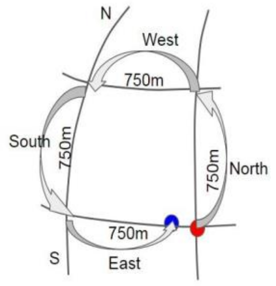
Our learning agent assumes the form of an electric vehicle that maneuvers within the Google Maps environment, making navigation decisions through various actions such as moving north, east, south, or west. The choice of actions can be influenced by the Double Deep Q-Network (Double-DQN) algorithm or selected randomly. Throughout the learning process, the agent embarks on initial explorations by navigating randomly across the map. Gradually, the reliance on random actions diminishes, and the agent leans towards opting for actions with the highest q-values generated by the Double-DQN model. The discount factor γ is set at 0.9, and Nb is configured as 32.

The Neural Network Architecture

The neural network architecture encompasses an initial input layer, which receives the geocode of the current position normalized by division with 180. This input layer is succeeded by two fully connected layers, comprising 10 and 6 neurons respectively, with ReLU activation functions and a dropout rate set at 0.25. The final layer comprises four dimensions, representing the q-values for each potential action. We initiate two identical networks: the Q-network, wherein the agent determines the action based on the present state, and the Target-network, which serves as a goal for the Q-network to attain. Weight updates are executed solely through backpropagation using the AdamOptimizer with a learning rate of 0.0001 in the Q-network at every step. The Q-network's weights are periodically copied to the Target-network every five steps.



(a)



(b)

Environment

The experimental setup is devised to assess the agent's capability in identifying an optimal route with minimal energy consumption and acceptable travel duration, employing the Double-DQN algorithm. A comparative analysis is conducted against the routes proposed by Google Maps API, which encompasses the Geocoding API, Directions API, and Elevation API. The experiment aims to ascertain if the Double-DQN algorithm enhances the agent's route selection proficiency compared to the routes suggested by the Google Maps API.

Interact with Google Map API

To initiate interaction with the Google Maps API, we begin by specifying both the starting position (which can be in the form of a place name, address, or geocode) and the destination position. These inputs are then utilized with the Geocoding API to retrieve the corresponding geocodes for these locations. Subsequently, utilizing the acquired geocodes, we create a rectangular boundary for the grid map within which our agent is permitted to navigate. Figure 5(a) illustrates the grid map and its symbolic representation. It's important to note that while each grid within the map is conceptually rectangular, the spherical geometry of Earth and our constraint on stride length introduce some deviations, as depicted in Figure 5(b).geometry and our restriction on the length of the stride which is demonstrated in figure 5(b).

Directional Choices and Navigating Instructions:

The agent's navigational choices encompass four cardinal directions: north, east, south, and west. Each arrow denotes the agent's movement from the current position (s) to the subsequent position (s0), covering a displacement of 750 meters within the grid map. However, the actual distance traveled by the agent might exceed or match 750 meters, contingent on the specific route provided by the Directions API.

Height and Elevation Computation:

To enhance the navigational experience, we extract the elevation information for each position within the navigating instruction list. Utilizing the geocodes from the instructions, we access the Elevation API to retrieve the height of each position. We then compute the elevation changes between consecutive positions. For instance, in Figure 7, the elevation variation between A (position 1) and 1 (position 2), between 1 and 2, between 2 and 3, and between 3 and B is determined. For simplicity, the height data for midpoints along the road, indicated within each instruction, is omitted.

This approach allows us to create a more accurate representation of the terrain and elevation changes encountered by the agent during navigation.

Consider Figure 6, where the agent is positioned at point A and intends to move southwards to point B. The route dictated by the Directions API follows highway 395, leading to a distance greater than 750 meters. To glean the navigating instruction list, we input the geocode of A and B into the Directions API, which returns guidance on each step of the route. The instruction list takes the form of pairs: {geocode of A, duration from A to 1, distance from A to 1, geocode of 1}, {geocode of 1, duration from 1 to 2, distance from 1 to 2, geocode of 2}, and so on. In this list, A, 1, 2, 3, and B correspond to the labeled points in Figure 6. The total number of instructions is contingent on the Directions API, with our example in Figure 6 containing four instructions from A to B.

Energy Consumption

We evaluate each action based on the energy required to travel with 750m displacement on the grid map (real distance is more than 750m). To compute the energy required between position A and position 1 in figure 6, for example, we use the duration and distance between position A and position 1 to calculate the average velocity V. Combine V with the elevation, we can get the angle θ of the road and consider the height of the road as linear increasing or decreasing shown in Figure 7. Because we don't take regenerative braking into account in our experiment, we treat the downhill road flat. In figure 8, we demonstrate how do we calculate the power required(Garcia-Valle and Lopes angle θ (degree) under velocity V (m/s) is as follows

where P (W) is power, f_m is the mass factor, M (kg) is the overall mass, g (m/s^2)is the acceleration of gravity, C_{rr} is the coefficient of rolling resistance between tires and road surface, ρ is the air density (kg/m^3), A (m^2) is the vehicle frontal area, C_d is the aerodynamic drag coefficient and V_W is the wind speed. We don't consider the early stage of acceleration, so α is zero. The parameters are listed in Table 1. Energy consumption should be computed by multiplying the power P by the duration.

Table 1: Parameters for power calculation

f_m	1.05
α	0 m/s^2
Mass	2000 kg
C_{rr}	0.02
ρ	1.225 kg/m^3
A	2 m2
C_d	0.5
V_W	0 m/s

Reward Arrangement The fundamental concept of defining the reward is based on the energy consumption in one stride from the current position to the next position, for example, from A to B shown in the figure 6. The energy is calculated by the method provided previous section. We then divide the energy by 10000 and times -1. In order to minimize the number of total steps during training, we add -0.1 to each transition if the next position is reachable. In other words, the reward r for taking any reachable step will be r = -0.1 - (energy consumption / 10000). If the next position is unreachable such as a lake or a river, r = -1 and the agent stay at the same current position and take the other action. If the distance of the next position and the destination position is less than the length of the predefined displacement (750m in the figure 6), the reward r for taking this action will become r = +1 - (energy consumption from the current position to the next position / 10000) - (energy consumption from the next position to the destination position / 10000). Noticed here +1 appears in the reward because of the success of this action which lead the agent to the destination.

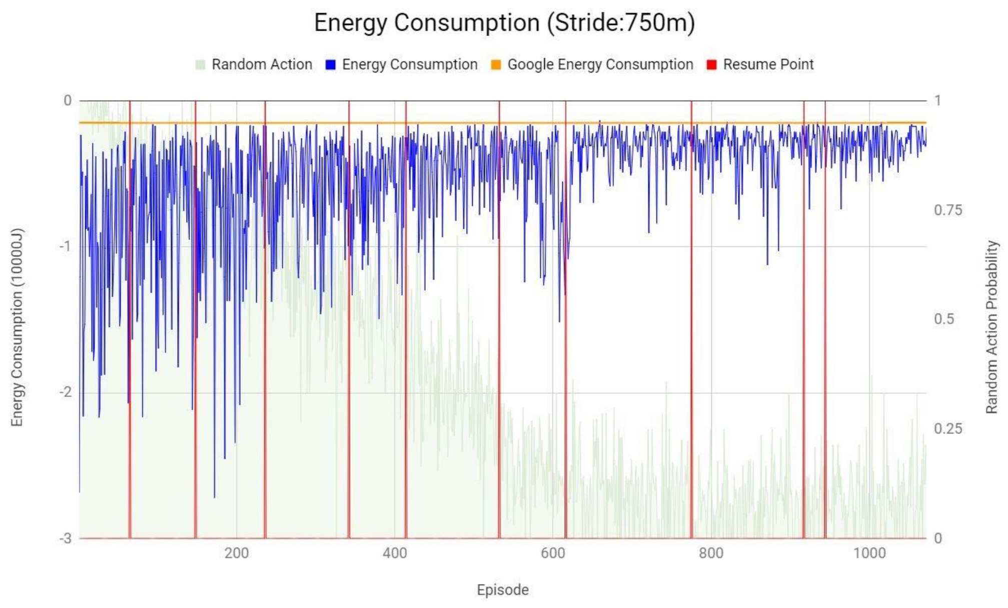


Figure 9: The energy consumption of the learning agent. The blue line is the energy consumed and the orange line is the energy required by taking the route which Google map recommend. The green line represents the probability of taking action randomly and the red line means we resume the training from previous training model

Energy Consumption Assessment:

Our evaluation of each action is rooted in the energy expenditure needed for the agent to traverse a 750-meter displacement within the grid map, despite the actual distance being greater. The computation of energy consumption is a multi-step process that involves several factors.

Results and Discussion

The training of the agent commences from the starting position identified by the geocode (40.4682572, -86.9803475) and proceeds towards the designated destination (geocode: 40.445283, -86.948429), employing a stride length of 750 meters. Episodes that span over 64 steps are deemed as unsuccessful. An important observation during training pertains to the intermittent blocking of our server by the Google Maps API. This disruption necessitates the termination of the training process and the subsequent resumption of the model from the point of interruption, as indicated by the red line in the graph. This unexpected issue leads to the depletion of the replay buffer, where actions are uniformly sampled to compute loss and update weights during the learning process. To address this, the model is resumed, and a phased reduction in the selection of random actions is employed to replenish the replay buffer, gradually diminishing the reliance on random choices.

Energy Consumption Analysis:

The energy consumption by the agent is depicted by the blue line in Figure 9. Notably, after approximately 600 episodes, the agent successfully identifies a path that minimizes energy consumption. The initial oscillations evident in the first 600 episodes are attributed to the heightened randomness in actions (illustrated by the green line) and the less precise Q-values provided by the Q-network. However, as the inaccuracies in Q-values are mitigated through minimized loss, the oscillations diminish, and energy consumption stabilizes.

Achieved Performance:

The agent achieves commendable performance in energy minimization, evidenced by the fact that after sufficient training, the energy consumption reaches a minimum value of 1327 joules (J). This optimal energy consumption is coupled with a corresponding travel time of 659 seconds. By comparison, the energy and time associated with the route recommended by Google Maps stand at 1489 J and 315 seconds, respectively.

These results underscore the agent's ability to identify efficient routes that lead to reduced energy expenditure, showcasing its potential to outperform the route suggestions provided by Google Maps in terms of energy efficiency.

Conclusion

In conclusion, the Double Deep Q-Network (Double-DQN) algorithm effectively demonstrates the capacity to learn from rewards and accumulated experiences. The experimental findings suggest that modifying the stride length to smaller distances, such as 20 meters or 50 meters, could potentially enhance both accuracy and energy reduction. However, further experiments are necessary to conclusively verify these potential gains.

It is important to note that the pursuit of increased accuracy in navigation comes at the cost of additional computational resources and time. The relationship between accuracy, computational demands, and time requirements underscores the trade-offs inherent in refining the learning process. Striking a balance between these factors is pivotal in optimizing the performance of the learning agent.

The success of the Double-DQN in navigating more energy-efficient routes underscores its potential to contribute to advancements in route planning and decision-making processes, with implications for practical applications in autonomous driving and other domains.

References

Simple Reinforcement Learning with Tensorflow Part 0: Q-Learning with Tables and Neural Networks

Value Iteration Networks

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, Pieter Abbeel

.

Deep Reinforcement Learning Based Dynamic Route Planning for Minimizing Travel Time Yuanzhe Geng, Erwu Liu, Rui Wang and Yiming Liu College of Electronics and Information Engineering, Tongji University, Shanghai, China

Wang, X., and Gupta, A. 2016. Generative image modeling using style and structure adversarial networks. In *European Conference on Computer Vision*, 318–335. Springer.