Reinforcement Learning on Route Planning through Google map for Self-driving System

Dung-Yi, Chao
Department of Mechanical Engineering
Purdue University
West Lafayette, Indiana 47906

Abstract

Self-driving system is a popular and important application of artificial intelligence. We will first introduce Double Q-Learning(Van Hasselt, Guez, and Silver 2016) which we implement their algorithm on our route planning system. Second, we will introduce value iteration network(Tamar et al. 2016) which might be the future algorithm for reinforcement learning. Third, we will go through Generative Adversarial Networks(Wang and Gupta 2016) which can be a creative ideal for next generation self-driving system. For the rest of the article, we will elaborate on the experiment environment, algorithm implementation and the result.

Introduction

Background knowledge

Reinforcement learning(RL) is a planning algorithm involved Markov decision process(MDP). We can think of it as mimic of human making decision. We call human as **agent**. In the daily life, human will encounter several situation such as seeing menu in the restaurant, taking a midterm exam or maybe fixing your own car. We call the daily life as **environment** and the situation as **state**. In a restaurant, we make a decision and order a meal. We call the term, order, as **action**. There is chances that the same meal which you have tried multiple times turns out to be ways too spicy than ever. We call the chance as **state transition probability**. It means that even the same action we make after a same state, the result can be different(spicy) from previous(normal flavor). During the meal, we will comment on this meal or restaurant based on the taste or the dinning experience. We say the taste or experience as **reward**. Reward can be positive, meaning that we really enjoy the food or the service. This catering experience would affect the decision we make next time when we are choosing among several restaurants or meals on the menu. This would gradually form or change the routine we make decision and we describe it as **policy**

The whole process can be simplified in Fig.1 and a few words as the following, under an environment, the agent encounter a state and make an action based on the policy. The environment would lead us to the next state and also give us a reward based on the action. We will make another action and receive another reward and enter the next state and so on. We will learn to form and modify the policy based on a series of state-action-reward procedure. Some important term and notation we will use in the following paragraph are specified here.

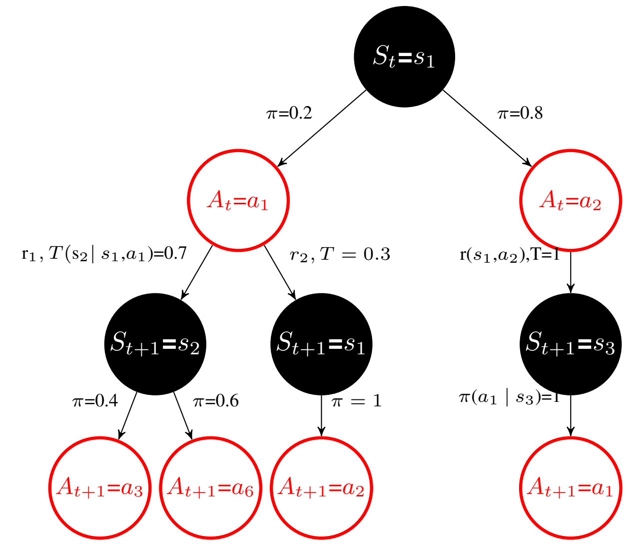

Figure 1: The procedure of iteration between state, action and reward.In state S_t, according to the policy π, we have 0.2 chance to choose action a_1 and 0.8 chance to choose action a_2. The reward r_1 and r_2 can be different.

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

Learning Process

Human learn from the feedback after we make some decision and action. We want the outcome of the ultimate goal as good as possible. For example, a five star restaurant is located on the top of a mountain. There are bad guys and ferocious animals on the way up to the top. We need to learn a good way to avoid them and get to the top rather than stop at the 2 star restaurant in the middle of the mountain.

Our agent learns from the reward for each action. We assign each action a value and call this action-value function **q(s,a)**. If the agent is at state s and with 4 choices of action , then we get four q value for this state s. The value are initially set to 0 and can be updated by the reward which we just encounter, furthermore, we can take the future reward into account in case we might be fooled by the current reward. Here is the functionality of discount factor γ. γ weights the importance of the future reward. If γ is near 1, it means the future reward is almost as important as the current reward and vice versa. We can express the combination of current reward and the future reward by the following equation:

$$q_\pi(s,a) =$$

$$r(s,a) + \gamma \sum_{s' \in S} T(s^{'}|s,a) \sum_{a' \in A} \pi(a^{'}|s^{'}) q_\pi(s^{'},a^{'})$$

We've already put the future reward inside the next state q value, $q_\pi(s^{'},a^{'})$. State $s^{'}$ represents next state relative to current state s. We are not sure which action $a^{'}$ is going to be selected before the agent really in state $s^{'}$, so we need to take expectation of $q_\pi(s^{'},a^{'})$ and that is where the summation and π take the role. Likewise, we are not sure what state $s^{'}$ will the agent enter after taking action a at state s, so we need $T(s^{'}|s,a)$ to represent the possibility and then apply summation to take all the chance into account. After the agent steps in most state and tries most of the action in each state, we can construct an instruction map to demonstrate the quality of taking a specific action given a specific state. At the end, the agent can choose the highest value of action at each state which would most likely to lead us to the optimum result and we refer it to greedy-policy.

We can update q(s,a) through off-policy or on-policy where the update algorithm will be specified in next section. What distinguishes off-policy and on-policy is how we determine the action $a^{'}$ of the next state $s^{'}$. For example, we bought a robot vacuum which was originally designed to turn right for 30% of the time, turn left for 60% of the time and move forward for 10% of the time in next state. If we let the robot to follow the policy we just mentioned, we refer it to on-policy. Conversely, off-policy is under the situation if we force the robot not to follow this policy but only guided by the direction which is detected with most dust.

Value function v_π(s) is a function to grade a specific state by taking all the q value in this state(implicitly with future state) into account which shows in the following equation:

$$v_\pi(\mathrm{s}) = \sum_{a \in A} \pi(a|s) q_\pi(s,a)$$

We will get more detail on how to learn q_π(s,a) in the next section.

Paper Survey

Deep Reinforcement Learning with Double Q-Learning

In this work(Hado, Arthur and David 2016), the author apply neural network on reinforcement learning and named it Double DQN. The functionality of neural network is to map the state **s** to **q**(s,·). The input of state **s** can be an n-dimensional vector such as an image. The output is a m-dimensional vector in which each element can be interpreted as **q** value corresponds to each action. In short, the neural network is a function to map from $\mathbb{R}^n$ to $\mathbb{R}^m$. We use θ to represent the parameter in the neural network.

We set up a goal or target for it to evaluate the **q** value computed by θ. The author used another neural network called target network governed by $\theta^{'}$ which contains the same architecture as the neural network governed by θ to compute the target. Now we get two neural network with identical structure governed by parameters θ and $\theta^{'}$ separately. $\theta^{'}$ is initially copied from θ. During training in one episode(agent begin from the start to the end), we will copy θ to $\theta^{'}$ for every N steps. In other words, we don't update $\theta^{'}$ within these N steps. The equation of target is $Y_t \equiv \mathbf{r}_{t+1} + \gamma \mathbf{q}(S_{t+1}, \underset{\mathbf{a}}{argmax} \mathbf{q}(S_{t+1}, \mathbf{a}; \theta_t), \theta^{'}_t)$.

Algorithm 1: Double DQN Algorithm

Input : D-empty replay buffer; θ-initial network parameter; $\theta^{'}$-copy of θ
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 **x** $\leftarrow$ ();
    for ($t \in \{$ *0, 1,...* $\}$){
        Set state $s \leftarrow$ **x**, sample action $a \sim \pi_B$;
        Sample next frame x^t from environment ϵ given (s,a) and receive reward r, and append x^t to **x**;
        if $|\mathbf{x}| > N_f$ **then** delete oldest frame $x^{t_{min}}$ from **x** **end**;
        Set $s^{'} \leftarrow$ **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
    }
}

We first input S_{t+1} into the θ network and choose the action **a** corresponds to the highest value in the output vector1 denoted by $\underset{\mathbf{a}}{argmax} \mathbf{q}(S_{t+1}, \mathbf{a}; \theta_t)$. At the same time, we input S_{t+1} into the $\theta^{'}$ network and get the output vector2. We get the **q** value from vector2 corresponds to action **a**. We get the target Y_t by combining current reward $\mathbf{r}_{t+1}$ and **q** value from target network $\mathbf{q}(S_{t+1}, \mathbf{a}; \theta_t^{'})$. The learning algorithm provided by Zyiu Wang's paper in 2016 is shown in Algorithm 1. 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.

Value Iteration Network

Value iteration network(VIN) is a model-free planning algorithm. We can use VIN with standard backpropagation and RL algorithm to deal with problems required visual perception, continuous control and natural language based decision. The goal is to learn a policy end-to-end which would generalize to solve different, unseen domain.

In Figure 2, the input ϕ(s) is an image(such as terrain image) and the current state. The output $\pi_{re}(a|\phi(s), \psi(s))$is a vector of probability over actions. f_R is basically a convolutional neural network(CNN) that transform the input image to a reward image $\bar{R}$(each pixel can represent a reward value). f_P is a state transition function $\bar{P}(s^{'}|s,a)$. $\bar{V}^*$ is a value function which has the same size of $\bar{R}$. Since the optimal policy at state **s** can depends only on nearby states which are a subset of the $\bar{V}^*$, the author use **attention** to achieve this purpose and outputs a vector ψ(s) which represent the value we really care about.

In Figure 3, it explains the heart of VIN, VI module, a mechanism to achieve the equation V(s) = $max_a R(s,a) + \gamma \sum_{s^{'} \in S} P(s^{'}|s,a)V(s^{'})$. Each iteration of VI module can be regarded as passing previous value function and reward image $\bar{R}$ into a convolution layer and max-pooling layer and then outputs a new value function $\bar{V}$. Each channel in the convolution layer corresponds to the Q-function for a specific action Q(s,a_1),Q(s,a_2),...Q(s,a_m) where m is the length of action set. The convolution kernel weights correspond to discounted transition probabilities $\gamma P(s^{'}|s,a)$. This layer is then max-pool along the actions channel to produce next iteration of value function $\bar{V}$, $\bar{V}_{i,j} = max_a \bar{Q}(a,i,j)$. We then attach $\bar{V}$ to $\bar{R}$ as it's second channel and feed them into convolution layer and max-pool layer K times to perform K iterations where K is the minimum value to convey the reward information from the goal to state **s**. After K iterations, the VI module will output $\bar{V}^*$ for the agent to make decision. We can then apply DQN or other RL methods to train parameters in Figure 2 and Figure 3.

Generative Image Modeling using Style and Structure Adversarial Networks(S^2-GAN)

Generative Adversarial Networks(GAN) contains two models: generator G and discriminator D. Generator G tries to generate images which looks like real image and discriminator D tries to distinguish between the real image and the image generated by G.

Structure(geometry of scene) and style(texture and illumination) are two key ingredients in image formation while ignored by most recent generative model. This paper decompose the generative process into two procedures: (i) use the Structure-GAN to generate a surface normal map and (ii)

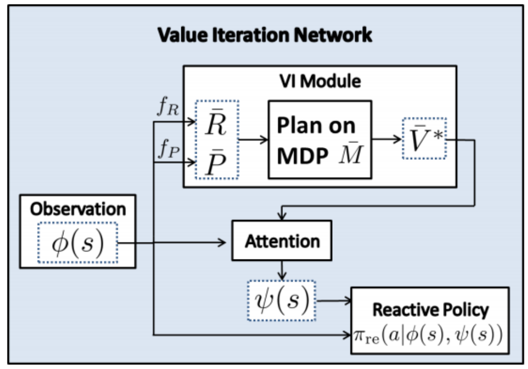

Figure 2: Value iteration network

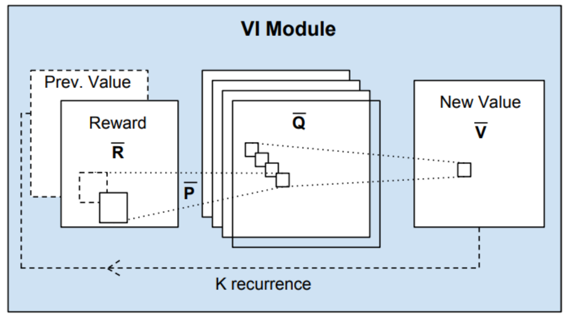

Figure 3: Value iteration module

Style-GAN to take surface normal map as input and generate the 2D image. The two GANs are trained independently and merged together via join learning.

Structure-GAN The input to the **G**enerative network is $\hat{z}$(100-d vector sampled from uniform distribution) and then generate the surface normal map(G($\hat{z}$): 72×72×3) in the end. The **D**iscriminator network takes image(72×72×3) as input and outputs a single value[0,1] which tells the surface normal is real(closes to 1) or generated(closes to 0).

Style-GAN The input to the **G**enerative network are $\tilde{z}$(100-d vector sampled from uniform distribution) and ground truth surface normal and then the network generates images(G(C_i,$\tilde{z}_i$)) with texture and illumination. The input to the **D**iscriminator network are ground truth surface normal, G(C_i,$\tilde{z}_i$), real image and real image's surface normal. This paper also includes fully convolutional network(**FCN**) which takes G(C_i,$\tilde{z}_i$) as input and estimates it's surface normal in order to make the **G**enerative network better by combining the loss in **FCN** and in **D**iscriminator network.

Joint learning for S^2-GAN After training the Structure-GAN and the Style-GAN independently, we are going to train both networks together shown in Figure 4 but first we remove the **FCN** part. Firstly, we input $\hat{z}$ and get generated surface normals G($\hat{z}$) and receive the first loss by feeding G($\hat{z}$) into the Discriminator network in Structure-GAN. Secondly, we input G($\hat{z}$) and $\tilde{z}$ into generator network of Style-GAN and get generated images(G(G($\hat{z}$),$\tilde{z}$)). We now receive

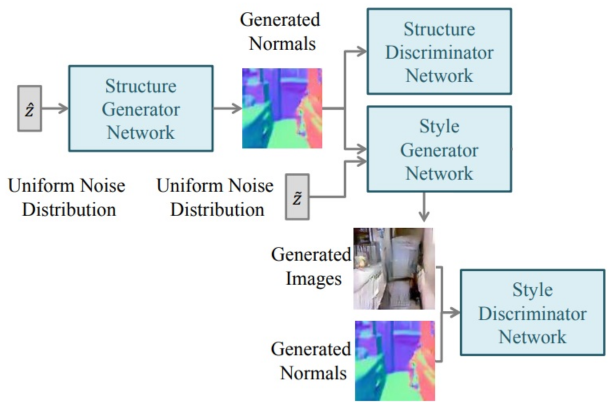

Figure 4: S^2GAN model

the second loss by feeding G(G($\hat{z}$),$\tilde{z}$) and G($\hat{z}$) into the Discriminator network in Style-GAN. We will combine the first loss and the second loss(scaled by 0.1) to train the generator network in Structure-GAN for producing better surface normals.

We can apply this technique to generate a heat map for the vehicle by input some parameters such as motor output, battery output, ambient temperature and so on. If the heat map is real enough, we can design a better cooling management system and minimize the number of heat sensor. The self-driving system can plan a better route based on the condition of the vehicle.

Experiments

In this section, we will first explain how do we set up the environment for the learning agent. Then we will introduce the algorithm which we implemented through tensorflow library. Lastly, we demonstrate the result of the experiment.

The Learning Agent

Our learning agent is an electric vehicle. It will navigate on the google map environment by choosing different action(north, east, south, west). The action can be determined by the Double-DQN or by random. During the learning process, the agent will first navigate on the map randomly, but we will gradually reduce the portion of choosing action randomly but adopt the action with highest Q value provide by the Double-DQN model.

The Neural Network Architecture The first layer is an input layer which takes in the geocode of the current position. The input layer is followed by one fully connected layer with relu activation function. The last layer is a output layer with four dimensions represent the Q value for each action.

Environment

In order to know if our agent is able to find the best route with minimum energy cost and acceptable duration under the Double DQN algorithm compared to the route provided by Google map API(Geocoding API, Directions API and Elevation API).

Interact with Google Map API We first specify the start position(could be place names, address or geocode) and the destination position and input to the Geocoding API to retrieve geocodes of these two position. We then use the start geocode and destination geocode as the two opposite corner to construct the retangle boundary of the grid map where our agent can only navigate on. The grid map and the symbols are shown in Figure 5. There are four directions choices(north, east, south and west) for the agent. Each arrow represents the agent navigating from the current position (s) to the next position ($s^{'}$) with 500 meters of displacement on the grid map. However, the real navigating distance of the agent will be larger or equal to 500m depends on the route provided by the Directions API. For example, in Figure 6, the agent is at current position denotes as A and heading south to the next position denotes as B. Apparently, the route provided by the Directions API is the route 395 and the distance is longer than 500m. We can get the navigating instruction list with the form: { 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 }.... where A, 1, 2 ,3 , B are shown in Figure 6 after we input the geocode of A and the geocode of B into the Directions API. The number of instruction is based on the Directions API and there are four instructions in our case of Figure 6 from A to B. We use each of the geocode in the navigating instruction list to get the height of each position from the Elevation API and compute the elevation within each instruction such as the elevation between A(position 1 in Figure 7) and 1(position 2 in Figure 7), the elevation between 1 and 2, the elevation between 2 and 3, and the elevation between 3 and B. We will omit the height in the middle of the road between two position provided within each instruction to simplify the complexity shown in Figure 7. The case for the the agent to travel from A to C will depends on the information that Directions API returns. If C is in a lake or somewhere unreachable, the Directions API will return false and the agent will regards the point C as a block. However, if C is considered reachable and the navigation instruction list returned by the Directions API is route 66, then we will analyze the elevation between position A, 1 and 4 even though the position 4 is not the identical position as C. We allow the route returned by the Directions API is located outside the grid map boundary while the agent should always navigate on the point within the boundary of the grid map.

Energy Consuming We evaluate each action based on the energy required to travel with 500m displacement on the grid map. 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 linearly increased or decreased 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 for a car with mass M (kg) to travel on the road with angle θ (degree) under velocity V (m/s) is as follows

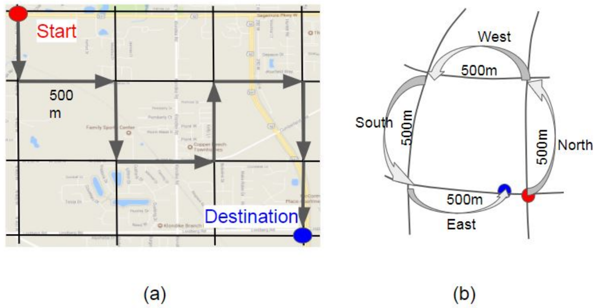

Figure 5: (a) Grid map of environment which the agent navigate on. The length and the direction of each arrow represents a certain length of displacement for a step and action taken by the agent. (b) In the real world, the agent will not be back to the same position if it take a round sequence of steps. This is caused by the sphere geometry.

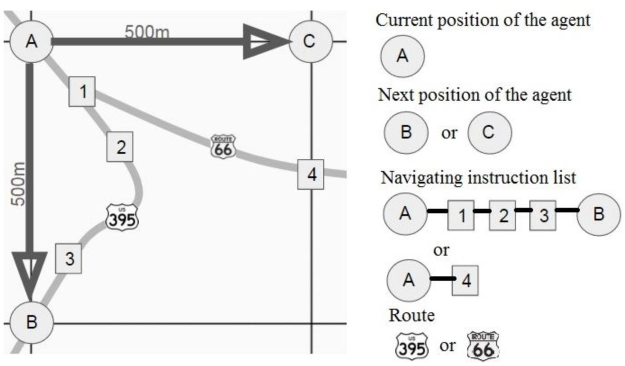

Figure 6: The graph only shows a part of the grid map. The agent can only navigate from one point to the other nearby point without crossing the boundary. We will analyze the road information such as road 395 if the agent travel from the current position A to the next position B.

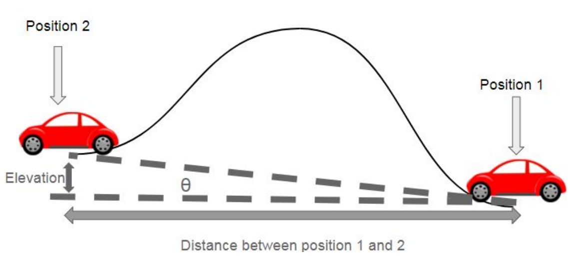

Figure 7: We only compute the elevation between position 1 and 2. In other words, the road height increases or decrease linearly whether there is turn on the road between the two position

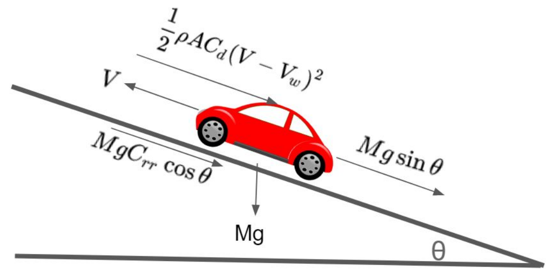

Figure 8: energy consuming

$$P = f_m M \alpha V + M g C_{rr} V \cos(\theta) + \frac{1}{2}\rho A C_d (V - V_W)^2 + M g V \sin(\theta)$$

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 m^2
C_d	0.5
V_W	0 m/s

Reward Arrangement The fundamental concept of defining the value of 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, energy consuming. 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 (500m 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.

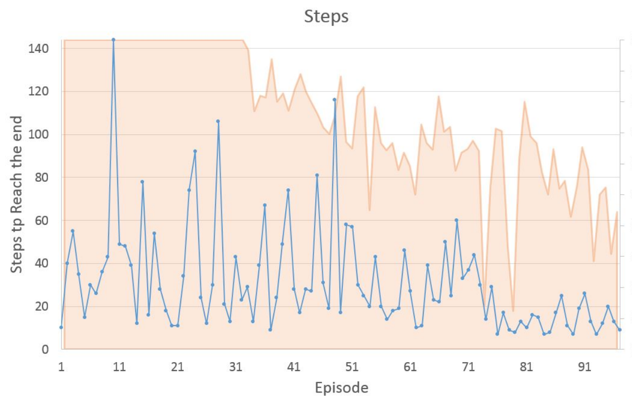

Figure 9: Steps to reach the destination on a straight road with each stride of 500m. The shaded area is the probability (right axis) of choosing action randomly

Electrical Vehicle Battery In the experiment, the battery performance will not affect the training process. The battery is able to carry totally 50000Wh of energy which is a standard offering by electrical vehicle manufacture, Tesla. In electrochemistry, it is recommended to use the the state of charge (SOC) from 90% $\sim$ 20% of a battery to improve it's life which we implement in our case. The SOC is calculated by the ratio between the current energy and the total energy. We will not take the battery degradation into the experiment. Further work can take the real factor on battery performance into account as part of the training process. For this experiment, we only demonstrate how much energy consumed and how many times the battery need to be charged in an ideal condition.

Results and Discussion

We first do a simple experiment for the agent to seek way from start point (geocode:40.4682572, -86.9803475) to the destination (geocode:40.4682572, -86.9507943252915) which is on the same road and same latitude. The result is shown in Figure 9. The y axis on the left hand side is the number of steps to reach the destination including the unreachable steps. The oscillation in the first 51 episode in Figure 9 is caused by the highly random action and inaccurate Q value provided by the Double-DQN. While the loss of inaccurate Q value is minimized, the oscillation is mitigated and the number of steps become less and stable.

We then train the agent on a more complicated map. The start position (geocode:40.4682572, -86.9803475) and the destination(geocode:40.445283,-86.948429) is on a diagonal. We also increase the stride length to 1000m to decrease the training time. Episode steps with more than 36 steps will be regarded as failed. Noticed that the result shown in Figure 10 is resumed from the training model after 88 episodes which the probability of random action is 0 and the agent reaches the destination successfully. During the training, google map api often blocked our server and we are forced to end the training process and resume the model from the interrupted episode. This problem will lead to the

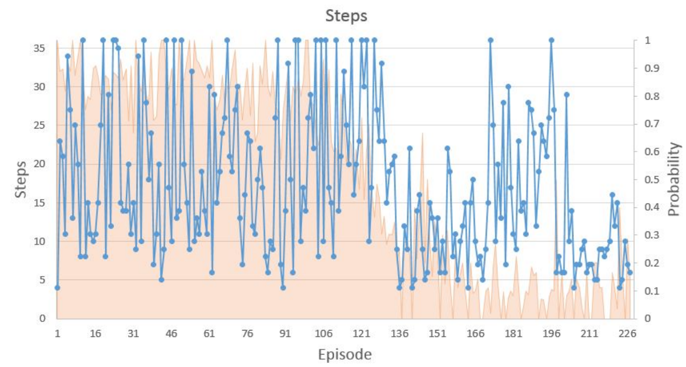

Figure 10: Steps to reach the destination on a diagonal with each stride of 1000m. The shaded area is the probability of choosing action randomly

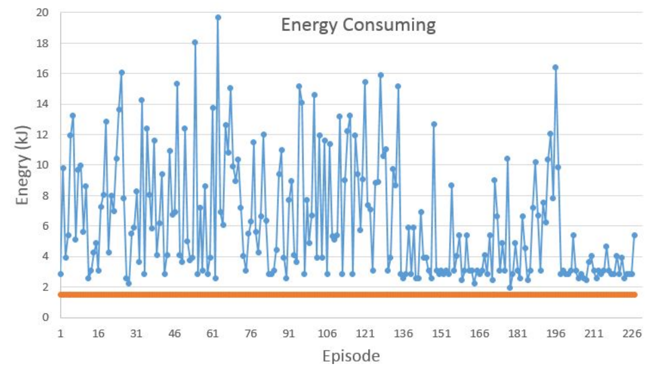

Figure 11: The blue line is the energy consumed by the agent and the orange line is the energy required by taking the route which google map recommend

empty replay buffer where we choose sample uniformly at random to compute the loss and updtate the weights during learning. As a result, we will need to resume the model and start choosing action randomly to refill the replay buffer and gradually decrease the portion of random action.

References

Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; and Abbeel, P. 2016. Value iteration networks. In *Advances in Neural Information Processing Systems*, 2146–2154.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *AAAI*, 2094–2100.

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