Machine Learning Engineer Nanodegree
Capstone Project

Ann Lee

November 25, 2018

Santa's Mission

I. Definition

Project Overview

Santa plans to give gifts to good children on Christmas Eve. He wants to know the shortest path and hopes to train the reindeer to complete the task themselves.

The project is mainly to apply Reinforcement Learning to solve the travel sales problems. The traveling salesman problem attempts to find the optimal solution for the shortest path. Reinforcement Learning learns in an interactive environment by repeatedly using its actions and experience feedback. As the model experiences more and more episodes, it will understand which behaviors are more likely to positive results. Reinforcement Learning is considered to be a solution to the general stochastic optimal control problem.

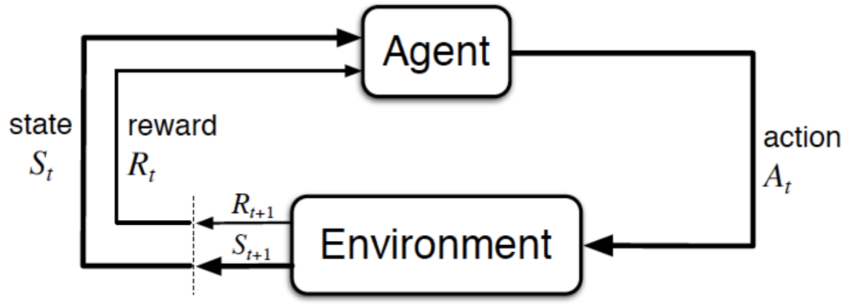
The dataset is a list of gifts consisting of latitude and longitude.

There are some researches mentioned to reinforcement learning can help the traveling salesman problem. Below are some related researches:
1. Reinforcement Learning: An Introduction 2nd Edition, Richard S. Sutton and Andrew G. Barto. Section 6.5, Q-learning: Off-policy TD Control.
2. Bello, Irwan, Pham, Hieu, Le, Quoc V, Norouzi, Mohammad, and Bengio, Samy. Neural combinatorial optimization with reinforcement learning. arXiv:1611.09940, 2016. Section 3, NEURAL NETWORK ARCHITECTURE FOR TSP.
3. Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, Le Song. Learning Combinatorial Optimization Algorithms over Graphs. arXiv:1704.01665, 2017. Section 6, Training: Q-learning.

Problem Statement

The requirement is to visit each location on the gift list and try to find the optimal

shortest path. There are two phases:
Step1: Reduce the number and range of the gift list.
Step2: Use the Q-Learning algorithm of Reinforcement Learning to find the optimal shortest path.



The agent selects the actions and the environment responds to these actions and presents the agent with a new status. Through this process to train and learn an optimal solution for the shortest path.

Metrics

This project requires to optimize the route Santa will take.

Visiting each location will send the gift to a specific place. After sending all the gifts on the gift list, it means completing the task.

And this project requires to visit each location on the gift list once and get the smallest route sum. Therefore, the metrics for this project can write as:

$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij}$$

where n is the number of gifts and cij is the distance from position i to position j.

The smaller the return value according to the above metrics, the shorter the path, the better the result.

Also, the reward is the negative distance between two selected positions. The measure of the total path is the minimum of the sum of the selected routes.

II. Analysis

Data Exploration

The dataset for this project is from "Santa's Stolen Sleigh" competition in the Kaggle. The dataset provides a list of gifts containing 100,000 items, each containing information of gift ID, latitude, longitude, and weight. In this project, only the latitude and longitude information are used. After processing the longitude and latitude, I can get the details knowledge of the destination location. For example,

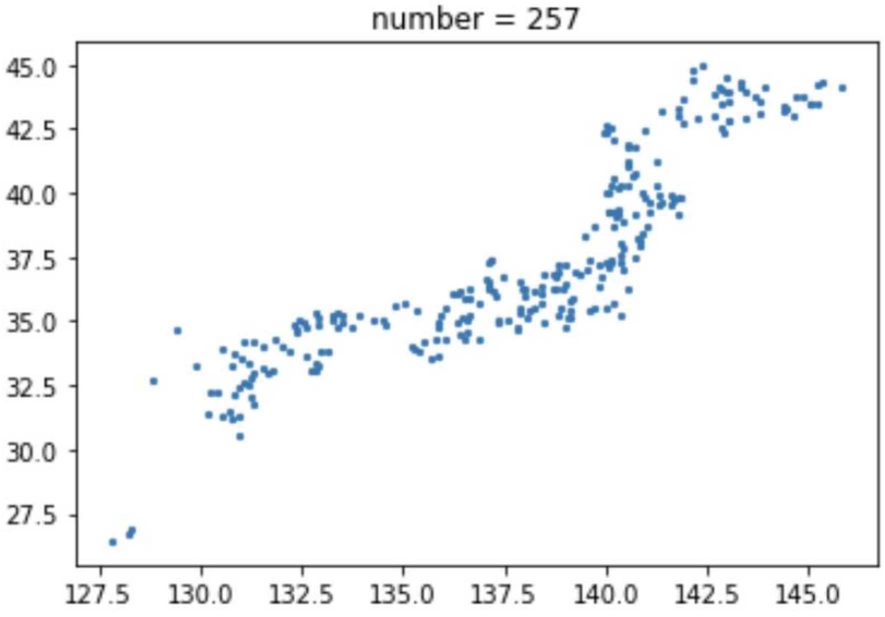
```
print(reverse_geocode.search([(34.02038529,135.233135)]))
[{'country_code': 'JP', 'city': 'Arida', 'country': 'Japan'}]
```

The gift list in the original dataset distributes throughout the world. However, since this project only focuses on the gift list of fewer items, the initial dataset will be narrowed down to a specific area. If the selected destination is Japan, there will be 257 locations in total. If the choice is Taiwan, there are 26 places to send gifts.

Each data means the location where the gift needs to be sent. And there is a path between any two positions.

The purpose of this project is to try to find the shortest path to complete the gift list.

Exploratory Visualization



The X-axis represents longitude, and the y-axis represents latitude. Each point denotes a location on a gift list. The above sample is an example of a list of gifts in

Japan. This gift list has a total of 257 gifts to be sent.

If there is a line connected in two points, it means that the path is selected. Adding all selected path lengths is the total distance of the route.

Algorithms and Techniques

Q-Learning of Reinforcement Learning is the selected algorithm.

Q-Learning is a specific temporal-difference algorithm. Temporal-difference methods update the value function after every time step, whereas Monte Carlo prediction methods must wait until the end of an episode to update the value function estimate. Besides, Q-Learning is off policy algorithm. Q-Learning directly learns the optimal policy. On policy, ex. SARSA learns a near-optimal policy while exploring.

In this project, the Q-Learning framework as follows:

- States: A state S is a sequence of actions on a graph G.
- Actions: An action A is a node of G that is not part of the current state S.
- Rewards: Minus of distance is the reward.

Q-Learning is a value-based Reinforcement Learning algorithm which is used to find the optimal action-selection policy using a Q function.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

- Action-selection policy: Probability of taking a random action in an epsilon greedy policy.
- α : This is a step-size parameter for the update step. It determines to what extent newly acquired information overrides old information.
- γ : This is the discount rate. Discount rate determines the present value of future rewards.

Benchmark

In this project, chose the greedy algorithm as the benchmark model. A greedy algorithm chooses the nearest unvisited location as next move. This algorithm quickly yields an effectively short route. Then through the shortest path to calculate the total distance of tours.

III. Methodology

Data Preprocessing

Because this project only focuses on a gift list with fewer items, the original dataset will narrow down to a specific country.

Reverse geocoding is the process of converting geographic coordinates into human-readable addresses. First, I used the geocoding API of Google Maps. However, it returned too much detail and was inefficient. After that, I switched to reverse_geocode, which was more straightforward and usable.

The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. I applied haversine to calculate the length of any two nodes. Also, considering the execution efficiency, I converted the result to an integer.

In the process and calculation phases, haversine takes a lot of time to calculate repeatedly, so I created a table to keep the distance information. Then I can look up the table to get the measurement results quickly.

Implementation

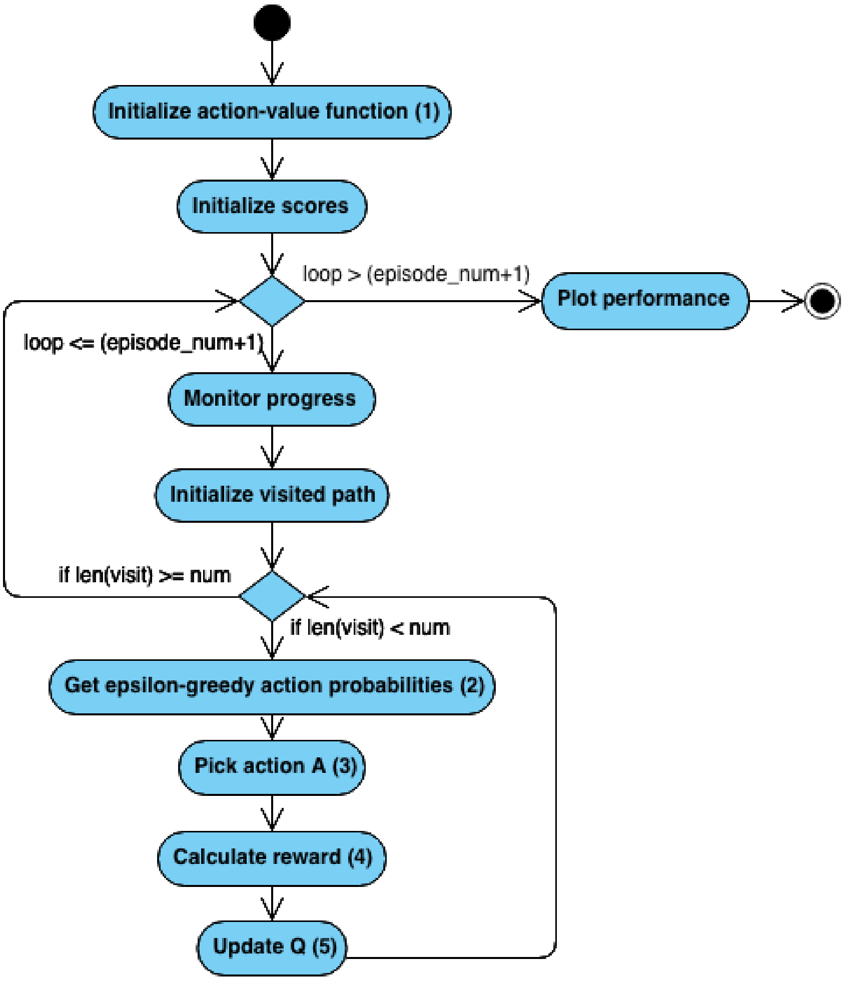
From the gift list data set, each data is a location to deliver a gift. And there is a path between any two sites.

There are two major functions:
q_learning(episode_num, alpha, gamma=1.0, dyna_num=None)
Parameters:

- episode_num: This is the number of episodes.
- alpha: This is the step-size parameter for the update step.
- gamma: This is the discount rate.
- dyna_num: Repeat times for Dyna-Q algorithm. This argument is for Dyna-Q algorithm only.

Return:

- Q: Action value function.



(1) Initialize action-value function

Q = defaultdict(lambda: np.zeros(num))

For each state s∈S and action a∈A, it yields the expected return if the agent starts in state , takes action , and then follows the policy for all future time steps.

(2) Get epsilon-greedy action probabilities

policy = epsilon_greedy_probs(Q[state], visit, e)

The detail description of function epsilon_greedy_probs is in below.

(3) Pick action A

action = np.random.choice(np.arange(num), p=policy)

The probabilities(policy) is the return value from the epsilon_greedy_probs function.

(4) Calculate reward

reward = -distance_table(visit[-1], action)

The reward is negative numbers of two-point distance.

(5) Update Q

```
visit.append(action)
next_state = ",".join(map(str, visit))

Q[state][action] = Q[state][action] + (alpha * (reward + (gamma * np.max(Q[next_state])) - Q[state][action]))
```

Next state is the current state adds the selected action.
Iteratively updates Q(s, a) using the Bellman Equation.

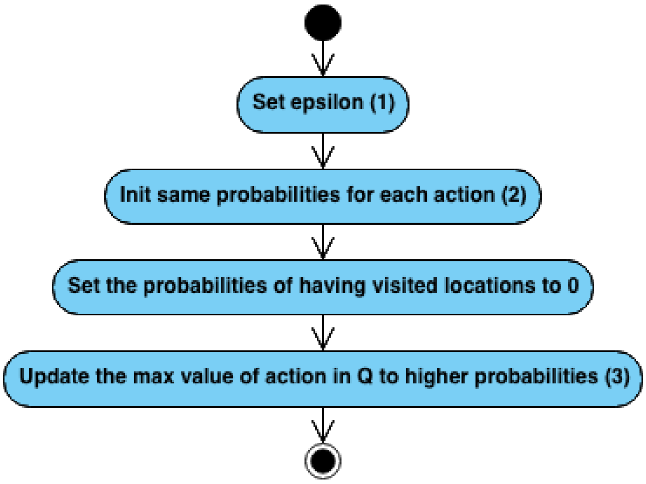
epsilon_greedy_probs(Q_state, visit, episode, eps=None)

Parameters:

- Q_state: Q[state]
- visit: It is a list which keeps the locations have visited.
- episode: This is the episode.
- eps: When using the Expected-Sarsa algorithm, epsilon is specified.

Return:

- Policy: The probabilities associated with each entry in Q[state].


Set epsilon (1)
Init same probabilities for each action (2)
Set the probabilities of having visited locations to 0
Update the max value of action in Q to higher probabilities (3)

(1) Set epsilon

```
epsilon = 1.0 / episode
```

In the beginning, the epsilon rates will be higher. High epsilon means explore the environment and randomly choose actions. Later the epsilon rate decreases and which means exploit the environment.

2) Init same probabilities for each action

```
policy = np.ones(num) * epsilon / (num - len(visit))
```

However, the probabilities of having visited locations are 0.

3) update the max value of action in Q to higher probabilities

```
policy[np.argmax(Q_new)] = 1 - epsilon + (epsilon / (num - len(visit)))
```

(1 - epsilon) adds the probabilities from (2)

In the implement process, I met an error:

```
np.random.choice: probabilities do not sum to 1
```

The parameter p of numpy.random.choice should sum to 1. But I update the probabilities of visited locations to 0. The error solved after changing the

```
policy = np.ones(num) * epsilon / num
```

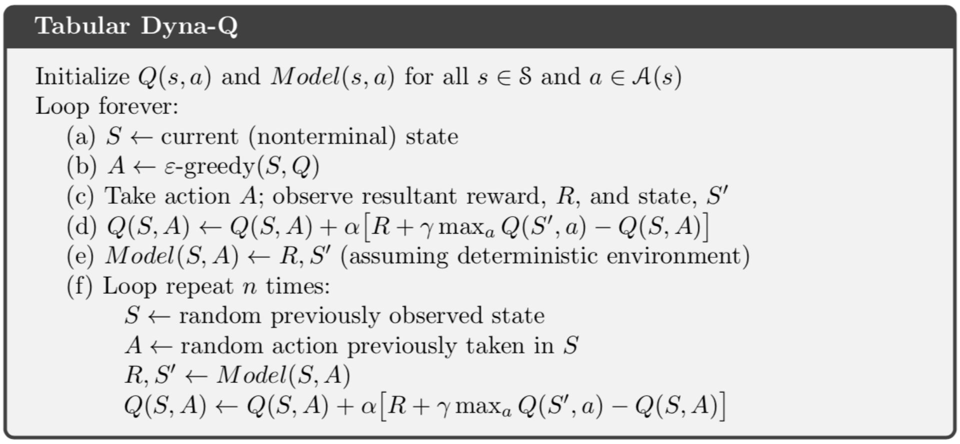
to

```
policy = np.ones(num) * epsilon / (num - len(visit))
```

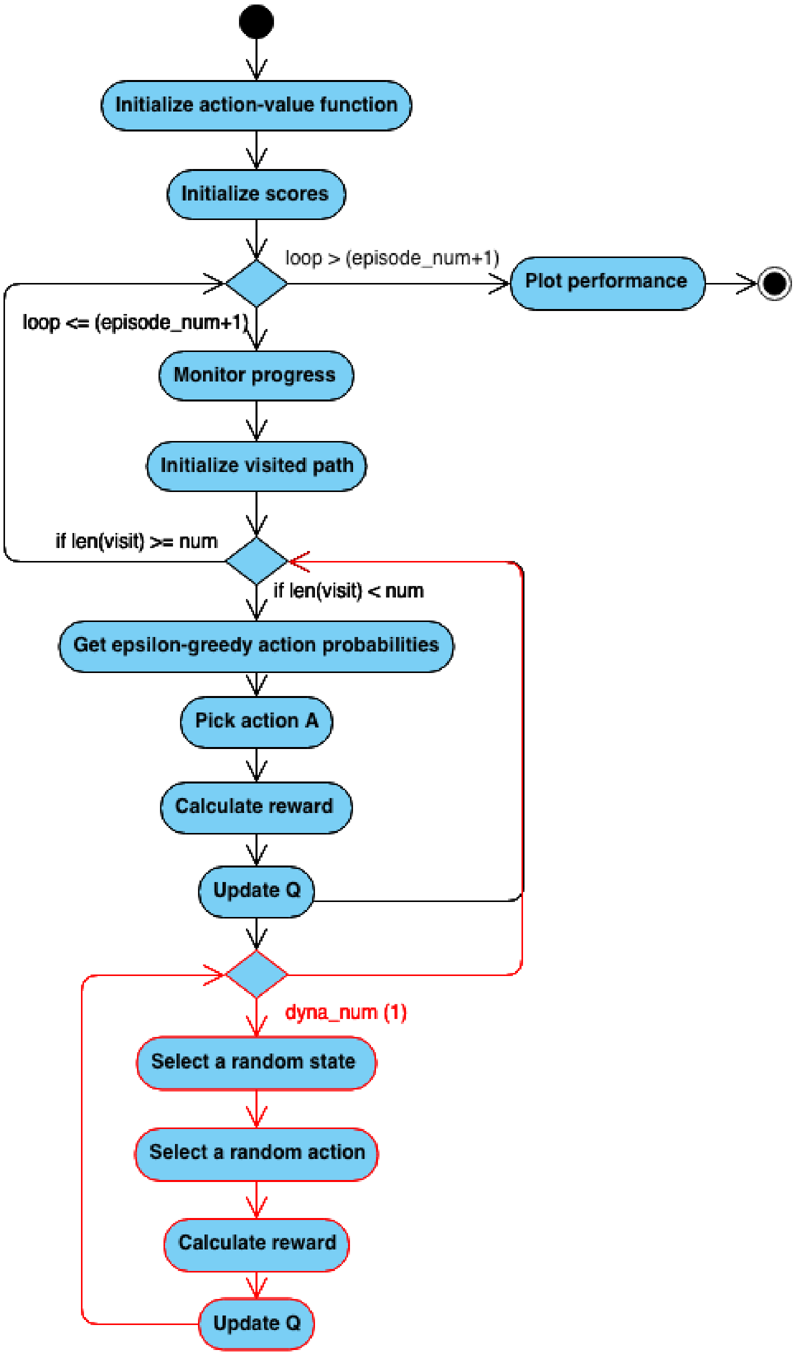
Furthermore, the output figure was mass and not easy to understand at the beginning. I change the output to the average value over every 100 episodes.

Refinement

Dyna-Q is the improvement algorithm.



```
Tabular Dyna-Q
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
    (a) $S \leftarrow$ current (nonterminal) state
    (b) $A \leftarrow \varepsilon$-greedy$(S, Q)$
    (c) Take action $A$; observe resultant reward, $R$, and state, $S'$
    (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
    (f) Loop repeat $n$ times:
            $S \leftarrow$ random previously observed state
            $A \leftarrow$ random action previously taken in $S$
            $R, S' \leftarrow Model(S,A)$
            $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
```



(1) Dyna_num

```
if dyna_num is not None:
    for _ in range(dyna_num):
```

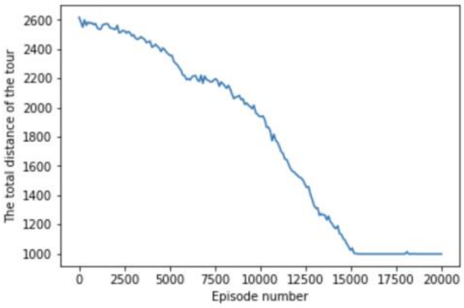
Loop repeat dyna_num times. The red lines part is additional actions from the Q-Learning algorithm.

Compare to Q-Learning and Dyna-Q, for example:

Q-Learning:

```
sarsamax = q_learning(20000, 0.01)
```

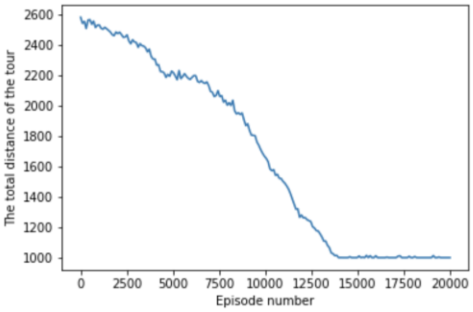
```
The shortest distance is 984 at 13211 episode.
```



Dyna-Q:

```
dyna = q_learning(20000, 0.01, 1.0, 100)
```

```
The shortest distance is 984 at 11842 episode.
```



The converge speed of Dyna-Q is faster than Q-Learning.

IV. Results

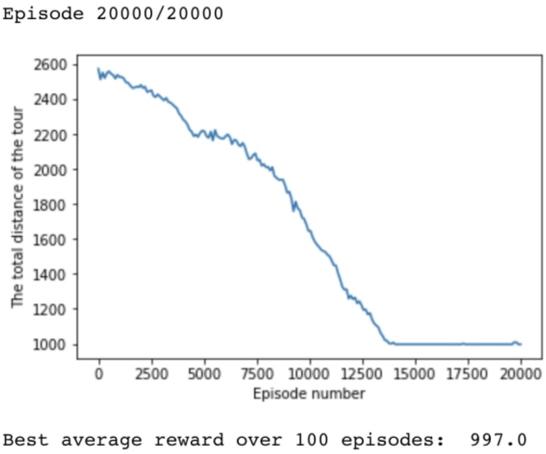
Model Evaluation and Validation

For the final model, I choose Taiwan with a list of 26 gifts as the basis.

(1) The parameters of final model:

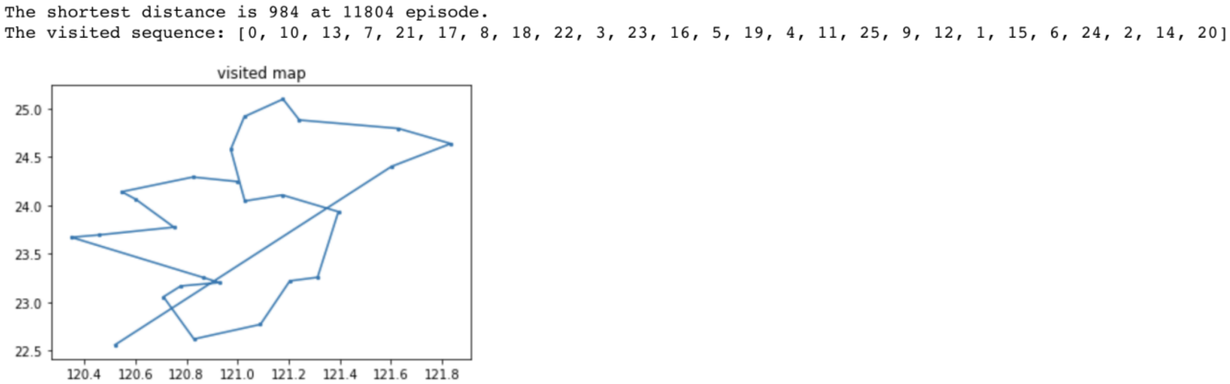
- episode_num: 20000
- alpha: 0.01
- gamma: 1.0
- dyna_num: 10

Converging from about 14,000 episode and the best average reward over 100 episodes is 997.0. The shortest distance is 984 at 11804 episode.



```
Episode 20000/20000
```

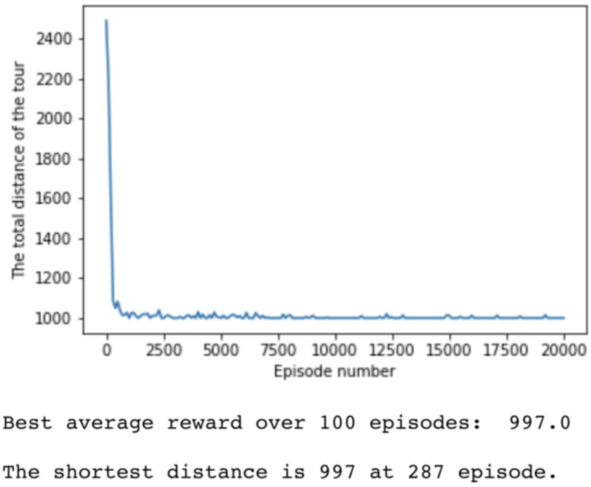
```
Best average reward over 100 episodes:  997.0
```



```
The shortest distance is 984 at 11804 episode.
The visited sequence: [0, 10, 13, 7, 21, 17, 8, 18, 22, 3, 23, 16, 5, 19, 4, 11, 25, 9, 12, 1, 15, 6, 24, 2, 14, 20]
```

(2) Set the alpha to 0.99 for comparison.

Although the result converges soon, the shortest distance is not as good as set the alpha to 0.01.

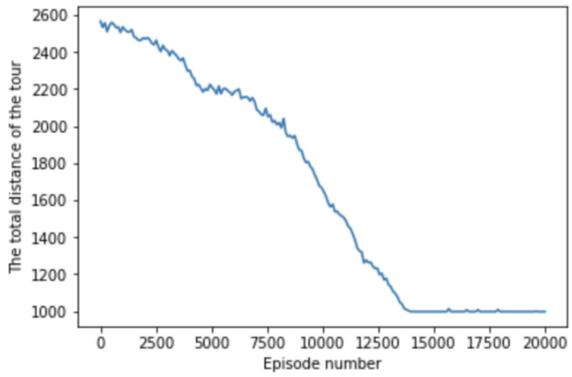


```
Best average reward over 100 episodes:  997.0

The shortest distance is 997 at 287 episode.
```

(3) Set the gamma to 0.9 for comparison.

The result is almost the same as setting the gamma to 1.0.

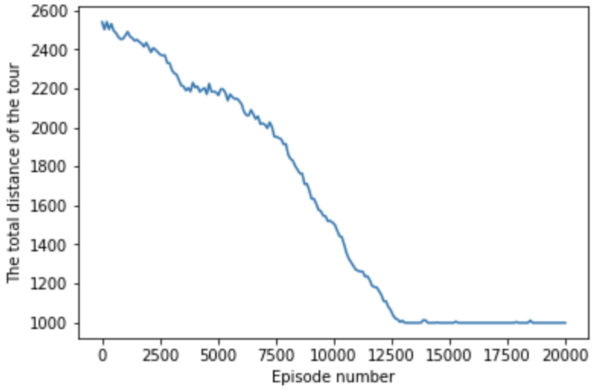


```
Best average reward over 100 episodes:  997.0

The shortest distance is 984 at 11836 episode.
```

(4) Set the dyna_num to 100 for comparison.

Performance is better than setting dyna_num to 10, but it takes a lot of time to execute.

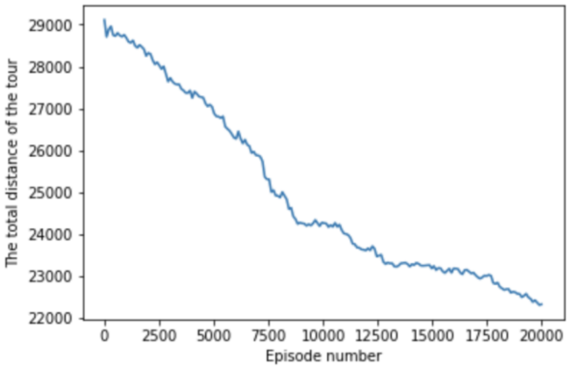


```
Best average reward over 100 episodes:  997.0

The shortest distance is 984 at 10879 episode.
```

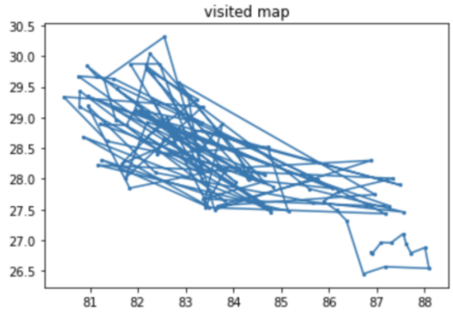
(5) Choose a list with 100 gifts for comparison.

For testing the robust, I select a longer list of 100 gifts. From the result below, it still needs to increase the number of episodes for better performance. But still can observe that it is learning and trying to converge.



```
Best average reward over 100 episodes:  22302.72
```

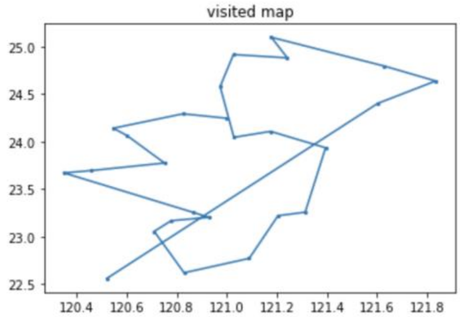
```
The shortest distance is 20790 at 19932 episode.
The visited sequence: [0, 13, 7, 33, 84, 86, 73, 68, 11, 16, 89, 77, 58, 66, 36, 56, 1, 2, 3, 4, 5, 6, 8, 9, 10, 12,
14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 35, 37, 38, 39, 40, 41, 42, 43, 44, 45, 4
6, 47, 48, 49, 50, 51, 52, 53, 54, 55, 57, 59, 60, 61, 62, 63, 64, 65, 67, 69, 70, 71, 72, 74, 75, 76, 78, 79, 80, 8
1, 82, 83, 85, 87, 88, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
```



Justification

The greedy algorithm is as the benchmark model. The result is below,

```
total distance: 997
visited sequence: [0, 10, 13, 7, 21, 17, 8, 18, 22, 3, 23, 16, 5, 19, 4, 11, 25, 9, 12, 1, 6, 15, 24, 2, 14, 20]
```

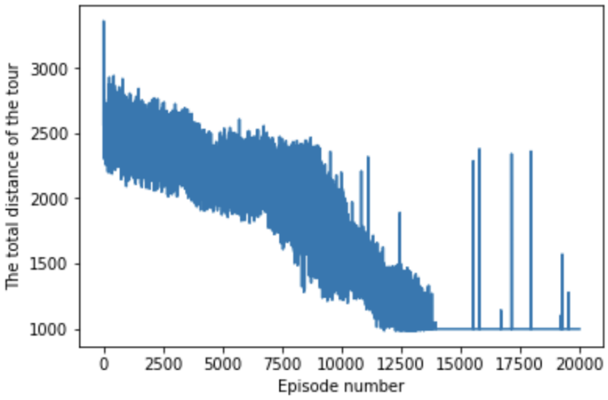


Compared to the final model, the final model converges to approximately 997 after running around 14000 episodes. However, the optimal total distance of the final model is 984, which is better than the 997 of the greedy algorithm. This solution should help Santa solve his problem.

V. Conclusion

Free-Form Visualization

The figure below shows the total distance change for each episode using the final model.



Although the result converges from around 14000 episode, there were some very high total distances after that which means the exploration continuous.

Reflection

The most interesting thing was that I had a story, and I wanted to use Reinforcement Learning to solve it, but I was not sure if it was possible.

Then I investigated and found some literature with a similar theme. But I was still not sure which algorithm I should choose. I carefully compared the details of the relevant algorithms and chose Q-Learning. But I still had no confidence that it could get good performance.

Then I ran into a problem. I initially chose Japan with 257 locations as my gift list. But even if I executed 50000 episodes, the result was terrible compared to the greedy algorithm of the benchmark model. I spent lots of time adjusting parameters and execution, but the results were unsatisfied. Even in the process of executing the Dyna-Q algorithm, the application memory is exhausted. Finally, I decided to change a gift list with a smaller range and fewer locations.

Fortunately, the final results were fit my expectations. It could use in general setting after fine tune.

Improvement

In this project, the calculation of the path is to visit each location once.

For more general traveling salesman problem, it should include a path back to the beginning.

```
*total += distance_table(visit[-1], start)*
```

Further, the starting node can be any point.

```
start = np.random.randint(0, high=num-1)
```

If there is strong machine support, Deep Q-Learning should have better performance.