[Project #2]
Traveling Salesman Problem

[01 팀]

	이름	학번	학년	E-mail	기여도(%)
팀장	선민준	20201273	3	sunminjun89@gmail.com	33
조원 1	남광규	20192898	3	ska00100@naver.com	24
조원 2	황준성	20221820	3	junseong5832@gmail.com	24
조원 3	홍진영	20221785	3	sunaep123@naver.com	19

1 서론

2 VALUE-BASED POLICY EXTRACTION

3 Q-LEARNING

4 DNN Q-LEARNING

5 DEEP Q-NETWORK

6 결론

1 서론

1.1 A* Tree Search 와 Greedy Search 를 통한 초기 경로 설정

A* 알고리즘, 그리디 알고리즘을 활용하여 초기 경로를 설정 하고, 이를 바탕으로 초기 Value Table 을 구성했다.

1.2 Value Function Initialization

초기 경로를 바탕으로 Value Function 을 초기화 하고, Value Iteration 을 통해 Value Table 을 업데이트 했다.

1.3 Policy Extraction

업데이트된 Value Table 을 통해 Policy Extraction 을 수행하여 초기 솔루션을 도출했다.

1.4 Q-Learning

Monte Carlo 와 Temporal Difference 기법을 활용하여 Q-Table 학습하고, 다양한 리워드 방식을 적용 했다.

1.5 DNN Q-Learning

신경망을 활용하여 Q-Table 을 더욱 정교하게 학습했고, 이를 통해 강화 학습의 성능을 향상 시켰다.

1.6 Deep Q-Network

DQN 을 활용하여 직접 Q-value 를 추론하고, 네트워크의 학습 성능을 비교 분석했다.

2 VALUE-BASED POLICY EXTRACTION

이번 장에서는 tree search 나 유전 알고리즘을 통해 구한 solution 을 바탕으로 value 함수를 초기화하고 초기화된 value 함수를 통해 value iteration 작업을 수행한 후 value table 을 구성한다. 구성된 value table 을 기반으로 Q-table 을 구성한 뒤 policy-extraction 을 통해 나온 solution 을 도출하는 과정을 설명한다.

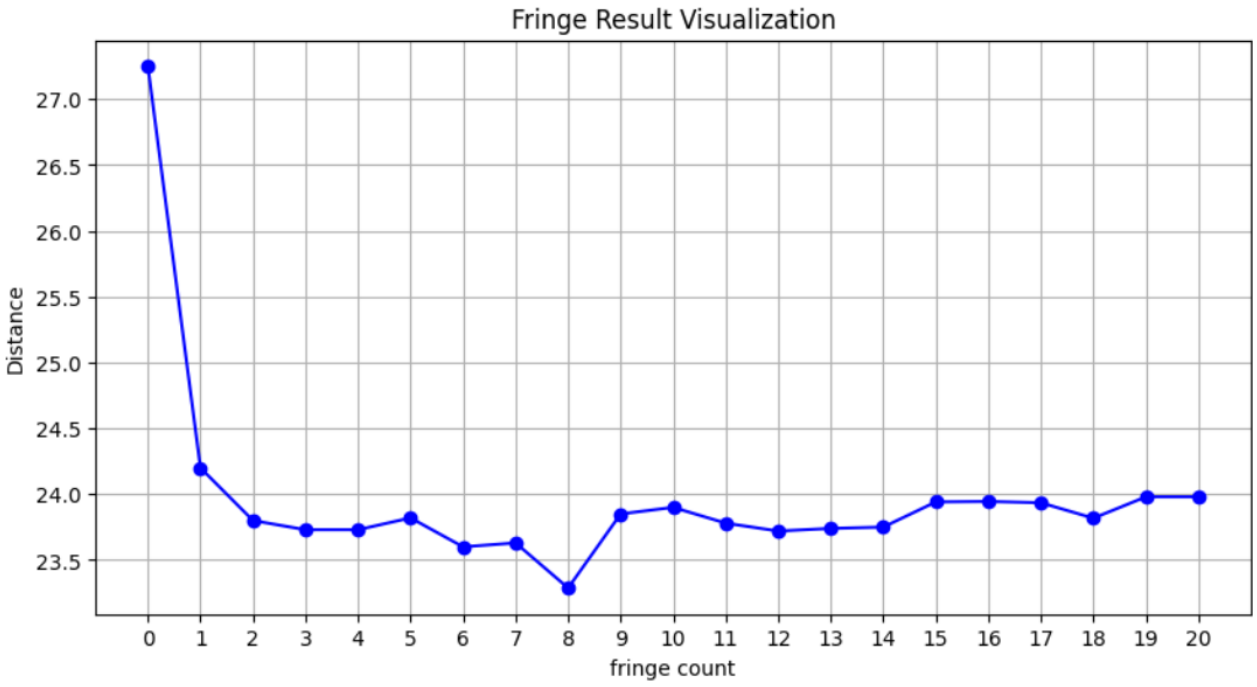
2.1 A* TREE SEARCH 를 통한 경로 추출(TREE SEARCH)

A* TREE SEARCH 는 시작 노드에서 현재 노드까지 도달하는데 드는 실제 비용과 현재

노드에서 목표 노드까지의 예상한 비용의 합으로 계산된다. A* 트리 탐색의 과정은 다음과 같다.

1. 현재 선택 가능한 후보들(fringe)결정: 모든 fringe 를 전부 탐색하는데에는 계산 비용이 너무 높다. fringe 를 제한해서 가장 결과값이 잘 나온 fringe 의 개수로 결과값을 도출한다.
2. heuristic 함수 결정: greedy 함수를 heuristic 함수로 사용하였다.
3. 인접 거리 행렬 활용: 각 도시에서 자기 자신 도시를 제외한 모든 도시에 대한 거리가 계산된 완전 연결 그래프를 사용하였다.
4. 정렬된 인접 거리 행렬 활용 : 트리 탐색시에 시간을 줄이기 위해 각 도시에서 자기 자신 도시를 제외한 도시들과의 거리를 오름차순으로 정렬한 2 차원 행렬을 활용하였다.

fringe 의 개수를 0 부터 20 개까지 fringe 를 제한해서 각각 A* 트리탐색을 진행한 결과 fringe 의 개수를 늘릴 수록 결과값이 좋게 나올거라 예상했지만 거리 결과값이 점차 감소하는 결과는 나오지 않았고 미묘한 거리변화만 나타났다. 따라서 fringe 의 개수를 8 개로 제한했을 때 나온 거리인 23.287351128928023 이 가장 결과가 좋게 나왔기 때문에 이때의 solution 으로 초기 경로를 활용한다면 다른 경로들로 구한 q table 에 비해 더 좋은 q table 이 나올꺼라 예상하였다.



<그래프 1> fringe 에 따른 결과값 비교

2.2 VALUE 함수 초기화(VALUE FUNCTION INITILIZATION)

초기 value 함수를 초기화 하기 위해 도시의 개수 만큼 0 이 들어있는 V(s) 리스트를 만들었다. VALUE TABLE V(s)란 상태 s 에 도달했을 때 얻을 수 있는 기대 보상의 총합을 뜻한다. 주요 과정은 다음과 같다.

1. Discount factor: 미래에 얻을 reward 의 가치를 떨어트리기 위해 사용하였고 0.9 로 설정하였다.
2. Reward: 도시 S 에서 도시 A 로 이동할 때 거리의 음수값을 취해 계산하였다.
3. 인접 거리 행렬 활용: 각 도시에서 자기 자신 도시를 제외한 모든 도시에 대한 거리가 계산된 완전 연결 그래프를 사용하였다.
4. V(S) 계산: 2.1 에서 구한 경로를 바탕으로 각 도시에 도착했을 때의 기대 보상의 총합을 계산한다.

다음은 VALUE TABLE 의 각각의 도시에 대한 초기 계산 과정을 설명한다.value table 의 모든 값을 0 으로 초기화하는 방법 대신 2.1 에선 나온 경로를 기반으로 value 를 초기화하는 방법을 선택하였다. 2.1 에서 구한 fringe 의 개수를 0 일 때 즉 그리디 서치로 나온 경로부터

fringe 개수를 각각 다르게 한 경로들까지 여러 경로들로 value 초기화 과정을 거쳤다. 경로가 모두 다르기 때문에 각각의 경로에 대한 value table 의 초기값은 모두 다르게 나타났다.

다음은 value table 을 초기화하는 과정을 나타낸다.

2.1 에선 나온 경로가 도시 S 에서 시작해서 도시 S 로 끝난 경로인 S->S*->S**->S 가 나왔다고 가정한다. V(S)는 도시 S 와 도시 S* 사이의 거리 X Dicount factor 의 0 승 ,도시 S*과 도시 S** 사이의 거리 X Dicount factor 의 1 승 , 도시 S**과 도시 S 사이의 거리 X Dicount factor 의 2 승을 취한 값을 다 더한 후 음수가 곱해져서 계산된다.V(S*)은 도시 S*과 도시 S** 사이의 거리 X Dicount factor 의 0 승 , 도시 S**과 도시 S 사이의 거리 X Dicount factor 의 1 승을 취한 값을 다 더한 후 음수가 곱해져서 계산된다. V(S**)은 도시 S**과 도시 S 사이의 거리 X Dicount factor 의 0 승을 취한 값의 음수가 곱해져서 계산된다.

<fringe 의 개수 8 개롤 제한한 A* 트리탐색으로 구한 경로로 초기화한 value talbe 의 value 값>

State	Value
0	-0.177023
1	-0.269818
2	-0.202758
3	-0.222262
4	-0.269619
...	...
993	-0.324620
994	-0.192156
995	-0.333064
996	-0.543071
997	-0.535575

<그리디 트리 탐색으로 구한 경로로 초기화한 value table 의 value 값>

State	Value
0	-0.141459
1	-0.430236
2	-0.296836
3	-0.238609
4	-0.496736
...	...
993	-0.509440
994	-0.261749
995	-0.166724
996	-1.788741
997	-1.723066

2.3 VALUE ITERATION

초기화된 value 함수로부터 value iteration 작업을 수행한다. 각 value iteration 에서 사용할 식은 다음과 같다. V*(s) = max(Reward(s, a) + (discount factor x V*(s)))

2.1 과 2.2 에서 구한 인접 거리 행렬, discount factor, reward,초기화된 value table 을 사용하였다. 이때 reward 를 주는 방식으로 -거리, -1/거리를 사용해서 수렴시켜보았다

2.3 에서 reward 방식에 대해서 자세히 이야기 하겠지만 여기서 reward 주는 방식을 여러가지로 사용하지 못하였다.

이유는 지수함수나 로그함수 등 reward 를 주는 방식으로 구현해봤지만 value iteration 과정에서 수렴하지 않고 발산해버리는 경우가 생겨버렸다. 따라서 value iteration 과정에서는 -거리, -1/거리로 reward 를 주는 방식으로 결정하였다.

value iteration 과정은 다음과 같다.

1. value 값 변화: 각 iteration 마다 현재 도시에서 다음도시로 갔을 때의 reward 와 discount factor 에 기존 value table 이 가지고 있었던 다음도시의 value table 값을 곱한 값을 더한 후 그 값이 최대가 될 때의 값으로 현재 도시의 value table 값을 update 한다.
2. value 값 수렴: value table 에서의 value 값의 수렴 여부를 판단하기 위해서 이전 iteration 과 현재 iteration 의 가치 함수의 value 값 차이가 가장 큰 도시의 value 값이 특정 임계값(0.000001)보다 작으면 수렴했다고 판단한다.

reward 를 -거리로 설정한 후 value iteration 진행한 결과 48 번째의 iteration 과 49 번째의 iteration 의 value 값 차이가 가장 큰 도시에 해당하는 value 값이 임계값보다 작아서 49 번째의 value iteration 과정을 거친 후 value 값이 수렴했다고 판단했다. 아래는 reward 를 -거리로 설정했을 때의 value iteration 과정을 거친 value table 이다

< fringe 의 개수 8 개롤 제한한 경로를 바탕으로 value table 초기화 한 후 reward 를 -거리를 취해 value iteration 과정을 49 번 거친 후의 value talbe 의 value 값>

State	Value
0	-0.045077
1	-0.112813
2	-0.115298
3	-0.102575
4	-0.063036
...	...
993	-0.142133
994	-0.036198
995	-0.068532
996	-0.247733
997	-0.237656

<그리디 트리 탐색으로 구한 경로로 value table 초기화 한 후 reward를 -거리를 취해 value iteration 과정을 49 번 거친 후의 value talbe 의 value 값>

State	Value
0	-0.045077
1	-0.112812
2	-0.115296
3	-0.102575
4	-0.063035
...	...
993	-0.142132
994	-0.036198
995	-0.068530
996	-0.247732
997	-0.237655

위에서 구한 value table 과 마찬가지로 49 번째에 수렴했고 value table 의 값이 같다는 것을 확인할 수 있었다.

초기의 value 값은 달랐지만 value iteration 과정을 거친 후 값이 같게 나온 이유는 리워드 주는 방식이 같았고 두 개의 경로 모두 그리디한 과정으로 트리 탐색을 하는 과정이 포함 되어있기 때문이라고 추측했다.

다음은 reward 를 -1/거리로 설정한 후 value iteration 을 진행한 결과이다 이때에는 71 번의 iteration 의 과정을 거친 후 수렴하였다.

아래는 reward 를 -1/거리로 설정했을 때의 value iteration 과정을 거친 value table 이다.

State	Value
0	-7.243588
1	-6.950970
2	-6.857975
3	-7.100895
4	-7.027194
...	...
993	-6.959775
994	-7.171219
995	-7.032007
996	-6.791793
997	-6.791793

거리를 리워드로 준 방식과 마찬가지로 리워드 주는 방식이 같았기 때문에 그리디 탐색으로 구한 경로와 A* 트리탐색으로 구한 경로 모두 value table iteration 후 결과값, 수렴한 iteration 번호가 같았다.

2.4 VALUE TABLE 로 Q-TABLE 구성

2.3 에서 구한 value table 을 기반으로 Q table 을 구성하였다.

기존 value table 은 도시의 개수만큼 들어있는 1 차원 리스트이지만 q table 은 행과 열이 각 도시의 개수로 구성된 2 차원 행렬이다. Q-TABLE 구하는 과정은 다음과 같다.

1. reward 계산: Q TABLE 을 구성할 때 reward 를 주는 방식은 여러가지를 사용할 수 있다.

   시도해본 방법은 -거리, -1/거리, 1-정규화, 각 도시에 대한 가까운 도시순서대로 예를 들면 한 도시 이 도시를 제외한 나머지 도시들에 대해 3 등분 해서 거리 가까운 순서대로 -로그함수(거리), -거리, -지수함수(거리) , 또 다른 방법으로는 각 도시마다 나머지 도시들에 대해 거리의 평균값을 계산한 후 평균 거리는 중심 도시에 대한 원이 되고 이 원 안에 있는 경우 로그함수를 취해서 더 많은 reward 를 주고 이 원 밖에 있다면 지수함수를 취해 더 많은 패널티를 주었다.

2. Q VALUE 값 계산: reward 에 discount factor x 2.2 에서 구한 value table 의 상대 도시에 대한 value 값을 더한 값으로 현재 도시와 상대 도시간의 q value 값을 계산한다.

< fringe 를 8 개로 제한한 경로를 바탕으로 -거리를 reward 로 주고 value iteration 을 진행한 후 q table 을 구성할 때 거리평균값을 구하고 로그함수와 지수함수를 취했을 때를 reward 로 준 경우의 Q-TABLE 의 Q VALUE 값>

	0	1	2	3	4	5	6
0	0.000000	-1.674143	-1.693036	-1.521530	-1.591447	-1.807220	-1.954405
1	0.751873	0.000000	0.928102	-2.259489	-2.456157	-2.050416	0.585452
2	0.728866	0.930337	0.000000	-1.969534	-2.410347	-2.600776	-2.388773
3	0.989103	-2.268703	-1.980983	0.000000	1.064437	-2.361192	-2.713271
4	0.807257	-2.500957	-2.457382	1.028851	0.000000	-1.975151	-2.510489
..	...	...	...	...	...	...	...
993	-1.794135	-2.590781	-2.886121	-2.007270	0.982192	0.850751	-2.148225
994	1.218995	-2.182974	-2.041608	1.816543	1.516364	-2.059616	-2.436817
995	1.292359	1.492500	1.157578	-1.861200	-2.054498	-1.972165	-1.857277
996	-2.461600	-3.397863	-3.929046	-2.713681	0.378951	0.540164	-2.533260
997	0.364608	1.148132	1.032514	-2.602868	-3.041128	-2.715579	-2.259020

	7	8	9	...	988	989	990	₩
0	-1.852278	-1.724151	2.130563	...	-2.651006	1.764583	-2.065237	
1	1.575974	0.724878	0.699900	...	-2.054596	0.856603	-1.882945	
2	1.236890	2.332319	0.541724	...	-2.779092	0.572071	-2.548663	
3	-2.383531	-1.919637	0.799559	...	-3.634598	0.628964	-2.843269	
4	-2.776947	-2.447323	0.838066	...	-3.546046	0.653742	-2.593995	
..	...	...	...	...	...	...	...	
993	-3.011824	-2.956956	0.697928	...	-3.139453	0.612602	-2.180814	
994	-2.365568	-2.016309	1.087907	...	-3.348631	0.849973	-2.545846	
995	1.123205	0.956783	1.110891	...	-2.332111	1.255671	-1.984480	
996	-4.001024	-4.051389	0.187519	...	-3.644617	0.152406	-2.519013	
997	2.011986	0.846558	0.280377	...	-0.012404	0.357853	-2.409260	

	991	992	993	994	995	996	997
0	1.885117	-2.397623	-1.881484	1.226986	1.271251	-2.643990	-2.161999
1	0.926071	-3.230396	-2.617169	-2.114022	1.532354	-3.519291	1.035773
2	0.645027	-3.685748	-2.910273	-1.970420	1.199667	-4.048239	0.922390
3	0.654365	-2.580914	-2.042871	1.876283	-1.830560	-2.844323	-2.724440
4	0.635366	-1.934734	0.911005	1.540517	-2.059444	-2.132425	-3.198287
..	...	...	...	...	...	...	...
993	0.563524	1.258620	0.000000	0.651497	-2.248438	0.892480	-3.469607
994	0.858978	-2.265091	-1.784170	0.000000	-1.770461	-2.500747	-2.725872
995	1.390824	-2.909298	-2.314679	-1.741362	0.000000	-3.190322	0.685069
996	-2.380212	2.202369	0.987520	-2.310366	-3.029040	0.000000	-4.573424
997	0.410518	-4.203168	-3.383636	-2.544561	0.837282	-4.582493	0.000000

[998 rows × 998 columns]

그리디 탐색으로 구한 경로로 -거리를 reward 로 주고 value iteration 을 진행한 후 q table 을 구성할 때 거리평균값을 구하고 로그함수와 지수함수를 취했을 때를 reward 로 준 경우의 Q-TABLE 의 Q VALUE 값도 위와 같게 나왔다.

또한 reward 주는 방식을 한 도시를 제외한 나머지를 도시들을 거리가까운 순서대로 3 등분한 후 다르게 로그,일반,지수 값을 취한 결과 역시 같게나왔다. 이유는 해당 리워드를 주는 방식이 거리에 따라 패널티를 다르게 주는 같은 방식이기 때문이라고 추측했다.

< fringe 를 8 개로 제한한 경로를 바탕으로 -거리를 reward 로 주고 value iteration 을 진행한 후 q table 을 구성할 때 -거리로 reward 로 준 경우의 Q-TABLE 의 Q VALUE 값>

	0	1	2	3	4	5	6
0	0.000000	-0.554269	-0.567041	-0.449441	-0.485076	-0.611841	-0.724423
1	-0.493307	0.000000	-0.460107	-0.865740	-0.931961	-0.741160	-0.611340
2	-0.503844	-0.457871	0.000000	-0.722107	-0.912684	-0.983731	-0.936417
3	-0.397693	-0.874954	-0.733556	0.000000	-0.382630	-0.885274	-1.069820
4	-0.468914	-0.976760	-0.959719	-0.418216	0.000000	-0.702895	-0.988563
..	...	...	...	...	...	...	...
993	-0.602220	-1.013512	-1.127064	-0.742010	-0.410567	-0.453665	-0.824602
994	-0.324347	-0.834592	-0.765341	-0.240566	-0.264134	-0.745739	-0.957326
995	-0.304273	-0.304636	-0.387039	-0.662665	-0.748761	-0.701347	-0.670175
996	-0.924762	-1.294341	-1.445398	-1.056012	-0.703555	-0.602287	-0.998023
997	-0.707428	-0.388132	-0.424777	-1.012819	-1.150129	-1.027681	-0.877660

	7	8	9	...	988	989	990	₩
0	-0.668983	-0.579667	-0.163179	...	-1.188623	-0.212952	-0.798551	
1	-0.302544	-0.530608	-0.522533	...	-0.887177	-0.453927	-0.698680	
2	-0.376663	-0.175302	-0.603474	...	-1.243009	-0.586857	-1.023338	
3	-0.936326	-0.692416	-0.477734	...	-1.545742	-0.557170	-1.139059	
4	-1.096508	-0.945368	-0.461584	...	-1.518333	-0.544759	-1.042039	
..	...	...	...	...	...	...	...	
993	-1.181185	-1.140810	-0.523466	...	-1.381836	-0.565535	-0.857067	
994	-0.928363	-0.743802	-0.370643	...	-1.454381	-0.456614	-1.022164	
995	-0.407694	-0.438636	-0.363363	...	-1.038783	-0.321031	-0.755533	
996	-1.475302	-1.463793	-0.838630	...	-1.548796	-0.866773	-1.010914	
997	-0.237655	-0.479699	-0.768716	...	-1.066192	-0.715095	-0.963533	

	991	992	993	994	995	996	997
0	-0.187852	-0.979251	-0.689570	-0.316356	-0.325381	-1.107153	-0.880749
1	-0.422021	-1.298720	-1.039900	-0.765640	-0.264782	-1.415769	-0.500491
2	-0.545235	-1.438072	-1.151216	-0.694153	-0.344950	-1.564591	-0.534901
3	-0.540561	-1.058842	-0.777611	-0.180827	-0.632025	-1.186654	-1.134392
4	-0.550117	-0.744521	-0.481754	-0.239980	-0.753707	-0.869783	-1.307287
..	...	...	...	...	...	...	...
993	-0.587940	-0.421173	0.000000	-0.537135	-0.844099	-0.550730	-1.394302
994	-0.448370	-0.917480	-0.632475	0.000000	-0.597459	-1.046164	-1.134962
995	-0.280887	-1.187295	-0.910340	-0.568360	0.000000	-1.310633	-0.620883
996	-0.891577	-0.276992	-0.455690	-0.855783	-1.149351	0.000000	-1.686255
997	-0.678146	-1.575926	-1.308331	-0.953650	-0.468670	-1.695324	0.000000

< fringe 를 8 개로 제한한 경로를 바탕으로 -1/거리를 reward 로 주고 value iteration 을 진행한 후 q table 을 구성할 때 -거리로 reward 로 준 경우의 Q-TABLE 의 Q VALUE 값>

	0	1	2	3	4	5	6
0	0.000000	-2.310314	-2.262312	-2.892465	-2.391298	-1.811699	-1.755741
1	-2.249352	0.000000	-2.910068	-1.385271	-1.199289	-1.483339	-2.129536
2	-2.199115	-2.907833	0.000000	-1.680148	-1.225020	-1.111583	-1.332304
3	-2.840717	-1.394484	-1.691596	0.000000	-3.125169	-1.236576	-1.162229
4	-2.375135	-1.244088	-1.272054	-3.160755	0.000000	-1.567039	-1.259640
..	...	...	...	...	...	...	...
993	-1.821033	-1.198044	-1.080999	-1.631506	-2.882903	-2.496300	-1.524115
994	-3.564453	-1.465672	-1.615311	-6.837716	-4.878278	-1.473937	-1.302088
995	-3.832699	-5.025086	-3.633930	-1.845633	-1.501755	-1.570632	-1.916055
996	-1.171543	-0.939887	-0.849127	-1.129990	-1.602748	-1.842008	-1.247377
997	-1.540136	-3.590696	-3.218924	-1.178681	-0.971312	-1.063879	-1.426086

	7	8	9	...	988	989	990	₩
0	-1.936920	-2.112951	-8.903520	...	-1.565219	-6.188910	-1.682814	
1	-5.565030	-2.336705	-2.167496	...	-2.253900	-2.526091	-1.960940	
2	-3.998852	-11.316065	-1.857738	...	-1.491002	-1.912941	-1.288809	
3	-1.342317	-1.735974	-2.389372	...	-1.202238	-2.021999	-1.155614	
4	-1.141859	-1.250242	-2.481203	...	-1.222171	-2.071470	-1.264949	
..	...	...	...	...	...	...	...	
993	-1.060317	-1.034519	-2.163324	...	-1.337353	-1.989997	-1.556221	
994	-1.354354	-1.607053	-3.171165	...	-1.272561	-2.509729	-1.290340	
995	-3.581913	-2.924074	-3.243729	...	-1.831157	-3.740427	-1.791416	
996	-0.856185	-0.812185	-1.318614	...	-1.200073	-1.274469	-1.305230	
997	-8.541476	-2.627898	-1.442036	...	-1.774085	-1.553726	-1.372446	

	991	992	993	994	995	996	997
0	-6.914291	-1.444391	-1.908382	-3.556462	-3.853807	-1.353933	-1.713457
1	-2.676068	-1.083131	-1.224432	-1.396720	-4.985233	-1.061315	-3.703055
2	-2.030791	-0.983256	-1.105151	-1.544123	-3.591841	-0.968320	-3.329048
3	-2.049446	-1.329678	-1.667107	-6.777977	-1.814993	-1.260632	-1.300253
4	-2.011672	-1.972801	-2.954091	-4.854125	-1.506701	-1.768976	-1.128470
..	...	...	...	...	...	...	...
993	-1.875154	-4.421696	0.000000	-2.014515	-1.339761	-3.273866	-1.061051
994	-2.505162	-1.550604	-2.109856	0.000000	-1.928108	-1.437723	-1.299581
995	-4.234242	-1.182975	-1.406002	-1.899008	0.000000	-1.142353	-2.670933
996	-1.219850	-11.071391	-3.178825	-1.247342	-0.981071	0.000000	-0.893069
997	-1.615122	-0.904147	-0.975080	-1.118269	-2.518721	-0.902138	0.000000

이를 통해 reward 를 다르게 주는 방식에 따라 q table 의 값이 크게 차이가 나는 것을 확인할 수 있었다.

2.5 POLICY EXTRACTION(ARGMAX 로 가장 좋은 VALUE 선택)

0 번 도시에서 시작해서 argmax 를 통해 solution 을 계산한다.

fringe 를 8 개로 제한한 경로를 바탕으로 -거리를 reward 로 주고 value iteration 을 진행한 후 q table 을 구성할 때 거리평균값을 구하고 로그함수와 지수함수를 취했을 때를 reward 로 준 경우의 Q-TABLE 로 SOLUTION 을 계산한 결과 25.986574325018 의 거리가 나왔다. 처음 경로의 23.287351128928023 과 비교했을 때 q talbe 로 구한 거리값이 더 늘어난 것을 확인할 수 있었다. fringe 개수를 0 으로 제한한 경로로 바탕으로 위에 과정을 똑같이 진행해도 25.986574325018 의 거리가 나왔다. 이유가 무엇 때문인지 분석해봤을 때 q table 에서 reward 를 주는 방식을 넣어 q table 의 값을 설정했는데 해당 reward 가 value table 로 구한 값보다 너무 커서 value table 의 value 값이 거의 영향을 주지 못하고 그리디 하게 되는 q table 의 값이 설정되는 현상이 나타났다. 때문에 reward 를 빼고 value table iteration 에서는 -거리를 reward 로 하고 q table 을 구성하는 과정도 진행해보았다. 해당 q table 로 경로를 구했을 때 381.3821428016102 의 거리가 나왔다. 또한 v table iteration 에서 -거리로 리워드를 주고 q table 을 구성할 때 reward 를 각각 -1/거리 , -거리로 진행했을 때의 q table 로 구한 경로에 대한 거리는 각각 609.3615297501083, 28.916484580600407 이 나왔다.

또 다른 결과로는 v table iteration 에서 -1/거리로 리워드를 주고 q table 을 구성할 때 reward 를 각각 거리평균 후 로그, 지수 취한 방법, -1/거리 , -거리로 진행했을 때의 q table 로 구한 경로에 대한 거리는 각각 26.236641220887147, 609.2986612029022, 609.2986612029022 이 나왔다. 결과적으로 value talbe 과정에서 -거리로 리워드를 주고 q table 을 구성할 때 거리평균값을 구하고 로그함수와 지수함수를 취했을 때를 reward 만든 q table 이 가장 결과값이 좋게 나왔다. 따라서 25.986574325018 이 나온 q table 을 사용하여 아래의 Q-learning 과정을 진행하기로 하였다.

3 Q-LEARNING

이번 장에서는 Monte Carlo (MC)와 Temporal Difference (TD) 기법을 활용한 value learning 수행해서 최적의 value table 을 학습하는 과정에 대해서 설명한다.

크게 2 가지 part Monte Carlo (MC)와 Temporal Difference (TD) 기법을 활용한 value learning 수행한 과정을 설명한다.

3.1 TEMPORAL DIFFERENCE (TD) 기법을 활용한 VALUE LEARNING

TD 기법을 활용해서 Q-TABLE 을 학습하는 과정은 다음과 같다.

1. 시작 상태를 무작위로 선택한다
2. 행동 정책에 따라 다음 도시를 선택한다.
3. 다음 도시에서의 Q value 중 가장 큰 값을 선택한다.
4. 현재의 Q value 를 업데이트한다.
5. 타겟 정책 최적화

시작 상태를 랜덤한 도시로 하나 고르고 난뒤 행동정책을 따른다.

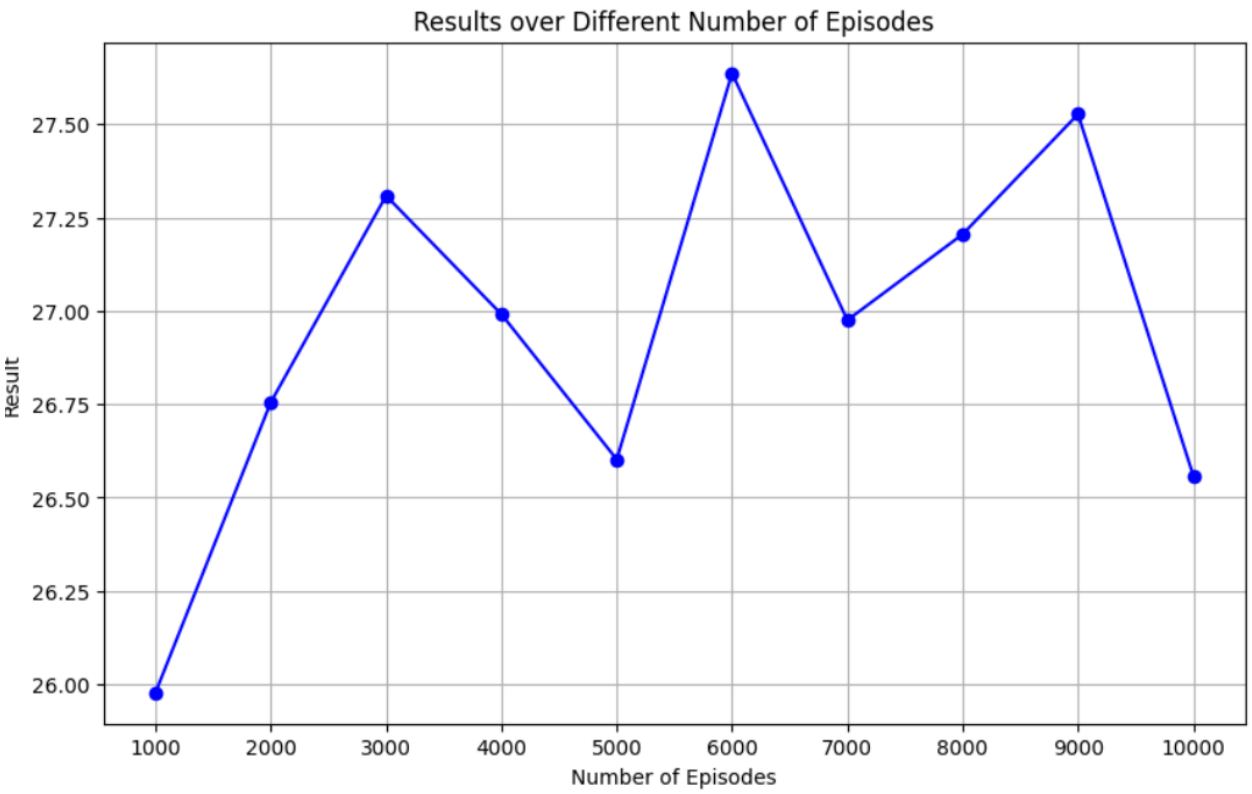
행동정책에서 epsilon 을 도입해서 0.1 로 고정된 값을 사용하였다.

랜덤하게 나온 확률값이 0.1 보다 작다면 무작위로 행동을 선택하는 탐험을 하였고 0.1 이상이 나온다면 가장높은 행동 즉 해당 상태에서 Q 값이 가장 높은 도시를 선택하였다. 이후 현재도시 선택된 다음도시 사이의 리워드를 계산하였다 여기서 사용된 reward 를 2.3 에서 사용한 reward 를 주는 방식을 사용하였다. 총 3 가지의 reward 를 주는 방식을 선택하였는데 -거리, -1/거리, 각 도시에 대한 거리 평균 구한 후 가까운 도시들 로그, 먼 도시들 지수취하는 방식을 선택하였다. 현재도시 ,다음 도시, reward 를 활용해서 q table 의 값을 업데이트해준다. 업데이트 할 때 사용된 식은 다음과 같다.

$$Q(s, a) = Q(s, a) + \text{learning rate}(\text{reward} + \text{discount\_factor} \times \max(Q(s' a')) - Q(s,a))$$

위의 과정을 여러 에피소드 과정을 거쳐서 반복한다.

아래는 reward 를 각 도시에 대한 거리 평균 구한 후 가까운 도시들 로그, 먼 도시들 지수취하는 방식을 선택한 후 에피소드를 각각 1000 번 2000 번 3000 번 4000 번 5000 번 10000 번 과정을 거치고 난 후 Q table 로 경로를 구한 결과이다.



에피소드 수가 늘어남에 따라 Q TABLE 로 구한 거리값이 더 좋아지기를 기대했지만 1000 번쨰의 에피소드를 제외하곤 에피소드 크기가늘어남에 따라 기존 25.98 보다 더 거리값이 늘어났음을 확인할 수 있었다.

아래는 에피소드를 1000 개로 여러번 했을 때의 결과이다.

```
calculate_distance(G, qpath)#에피소드 1천번
```

```
25.977642848941777
```

```
calculate_distance(G, qpath)#에피소드 1천번
```

```
25.97917188220382
```

같은 에피소드여도 epsilon 을 도입했기에 결과가 각각 다르게 나온 것을 확인할 수 있다.

또한 에피소드를 500 번 정도로 작게했을 때는 Q 값이 제대로 update 되지 않을 것을 확인할 수 있다.

```
calculate_distance(G, qpath)#에피소드 500
```

```
25.986574325018
```

3.2 MONTE CARLO (MC)기법을 활용한 VALUE LEARNING

MC 기법을 활용해서 Q-TABLE 을 학습하는 과정은 다음과 같다.

1.트리 서치를 활용해서 경로 일부분을 샘플링한다.

2. 시작 상태를 1 번에서 샘플링한 경로에 없는 도시안에서 무작위로 선택한다

3. 행동 정책에 따라 다음 도시를 선택한다.

4. 힌개의 에피소드를 만든다. 이때 한개의 에피소들 만들 때에는 현재도시, 행동정책에 따라 나온 도시 , 그 다음 도시등 모든 도시를 다 방문해야한다.

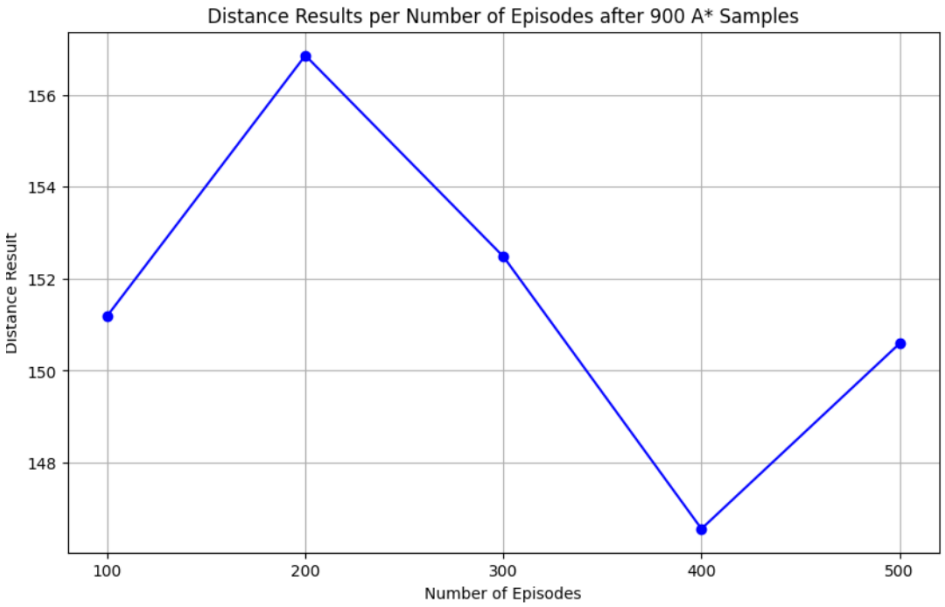
5. 여러번의 에피소드를 거친 뒤 q 값을 update 해준다.

한가지 예시를 들면 만약 0->1->2->3->0 번 도시로 이동했다고 가정할 때 0 번도시는 1,2,3,0 을 거친 모든 reward 를 저장하고, 1 번도시는 2,3,0 을 거친 reward, 2 번도시는 3 번,0 번을 거친 reward, 3 번도시는 0 번도시를 거친 리워드를 저장한다.

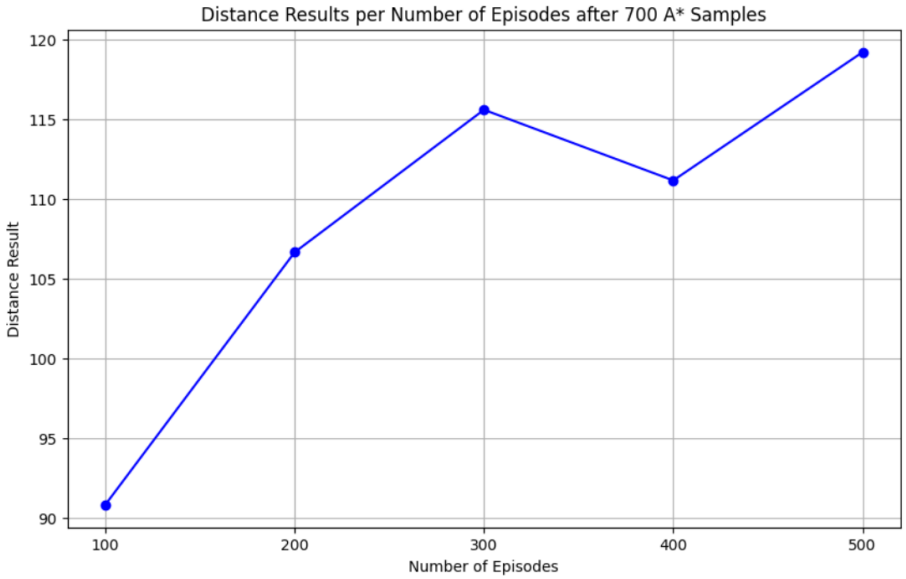
각각의 현재도시, 다음도시, total reward 를 바로 계산하지말고 따로 저장해둔다.

이후 여러번의 에피소드를 거친 뒤 현재도시 다음도시 전체 reward 정보중에 현재도시와 다음도시 겹치는 경우가 있다면 전체 reward 를 평균내서 q value 를 update 해준다.

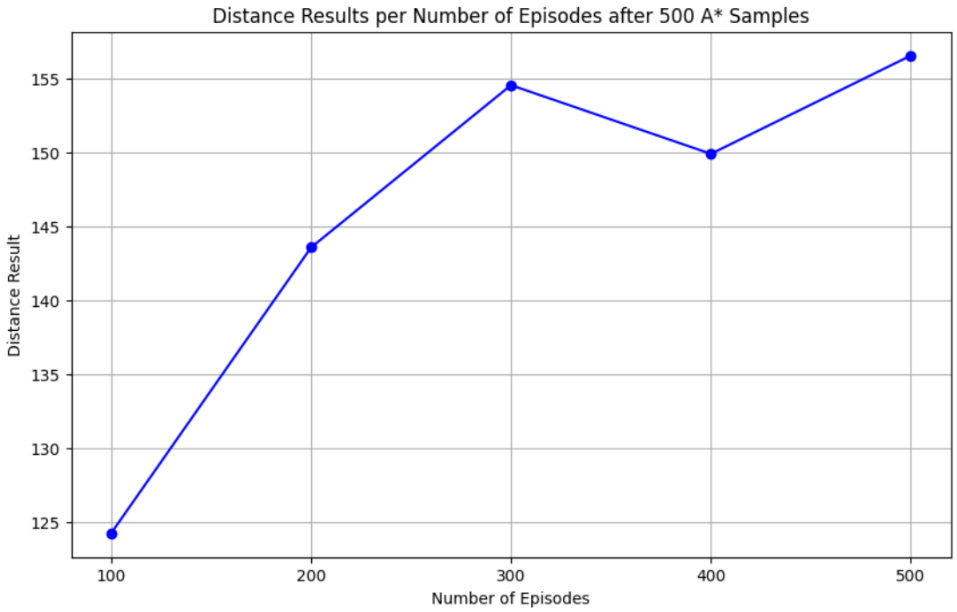
아래는 900 개의 도시들을 에이스타로 샘플링한뒤 나머지 도시들에 대해 몬테카를로 알고리즘에서 에피소드를 각각 100, 200, 300, 400, 500 로 실행했을 때의 결과이다.



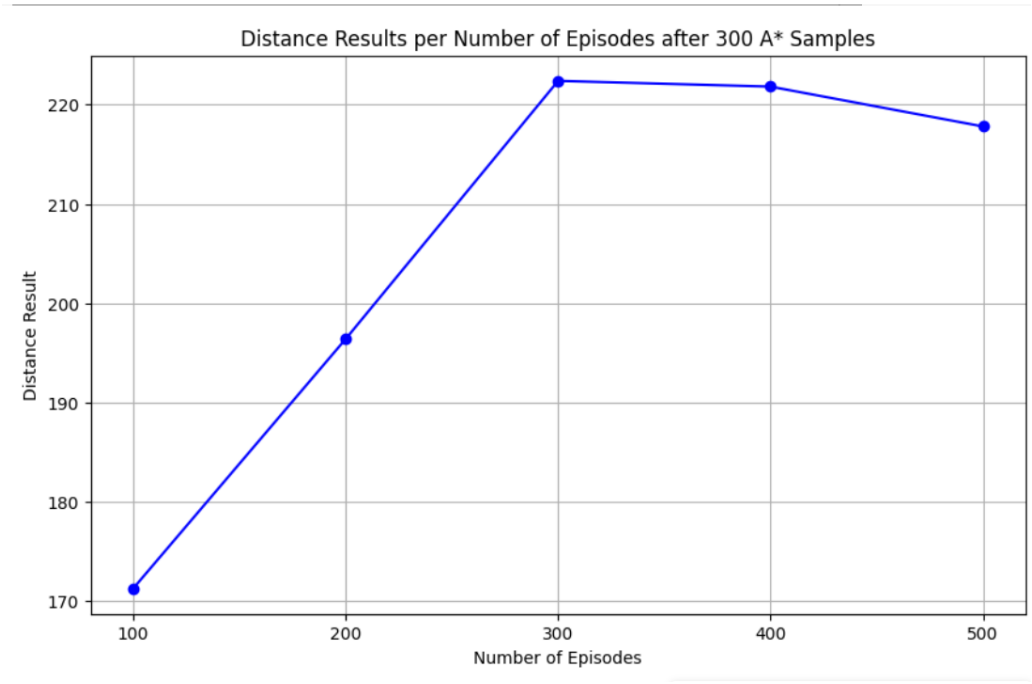
아래는 700 개의 도시들을 에이스타로 샘플링한뒤 나머지 도시들에 대해 몬테카를로 알고리즘에서 에피소드를 각각 100, 200, 300, 400, 500 로 실행했을 때의 결과이다.



아래는 500 개의 도시들을 에이스타로 샘플링한뒤 나머지 도시들에 대해 몬테카를로 알고리즘에서 에피소드를 각각 100, 200, 300, 400, 500 로 실행했을 때의 결과이다.



아래는 300 개의 도시들을 에이스타로 샘플링한뒤 나머지 도시들에 대해 몬테카를로 알고리즘에서 에피소드를 각각 100, 200, 300, 400, 500 로 실행했을 때의 결과이다.



TD 기법과 마찬가지로 샘플링 개수를 늘려서 진행했을 때 Q TABLE 로 구한 경로들의 거리값이 더 좋아지기를 기대했지만 에피소드 개수를 늘릴 수록 Q TABLE 로 구한 거리값이 더 늘다가 일정 개수가 지나면 Q TABLE 로 구한 거리값이 크게 바뀌지 않는것 을 확인할 수 있었다.

또한 아예 처음부터 tree search 와 같은 기법들을 사용하지 않고 0 번부터 무작위로 샘플링해서 나온 결과값이 384.25323270179891 가 나왔고 이는 최적에 가까운 Q TABLE 에서 학습 후 랜덤성이 과도하게 반영되었다고 판단했다.

또한 트리탐색을 하는 도시들의 수가 늘어날수록 Q TABLE 로 나온 거리 결과값이 예상과는 달리 700 개의 도시들에 대해 트리서치로 샘플링한 뒤 나머지를 MC 알고리즘을 돌려서 나온 거리 결과값이 900 개의 도시들에 대해 트리서치로 샘플링한 뒤 나머지를 MC 알고리즘을 돌려서 나온 거리 결과값보다 좋은 것을 확인할 수 있었다.

4 DNN Q-LEARNING

이번 장에서는 Deep Neural Network(DNN)를 활용하여 앞선 과정에서 추출한 Q-Table 을 더욱 발전시키는 과정을 설명한다. DNN 을 사용하여 Q-값을 더 정교하게 추론하고, 이를 통해 강화 학습의 성능을 향상시키는 방법을 단계별로 살펴본다.

4.1 Q-TABLE 학습을 위한 신경망 구축

DNN 을 통한 Q-Table 을 보다 정교하게 학습하기 위해 신경망을 다음과 같이 구성하였다. 이 신경망은 Fully Connected Layer 로 구성되어 있으며, 각 층의 세부 사항은 다음과 같다.

- 입력층 (Input Layer): 입력 차원은 PCA 를 통해 100 으로 압축된 distance matrix 이다.
- 은닉층 1 (Hidden Layer 1): 첫 번째 은닉층은 512 개의 뉴런을 가지며, ReLU 활성화 함수를 사용한다.
- 은닉층 2 (Hidden Layer 2): 두 번째 은닉층은 256 개의 뉴런을 가지며, 역시 ReLU 활성화 함수를 사용한다.
- 출력층 (Output Layer): 출력 차원은 Q-Table 의 크기에 해당하며, 최종적으로 Q-Table 을 추론하게 된다.

신경망의 입력은 distance matrix 를 PCA(Principal Component Analysis)를 통해 차원을 100 으로 압축한 것이다. 이는 원래 고차원 데이터를 저차원으로 변환하여 계산 효율성을 높이고, 중요한 특징을 추출하기 위함이다.

출력으로는 998 X 998 예측된 Q-Table 이 나온다. Q-Table 은 상태-행동 쌍에 대한 Q-값을 포함하며, 이는 에이전트가 최적의 행동을 선택할 수 있도록 하는 데 사용된다.

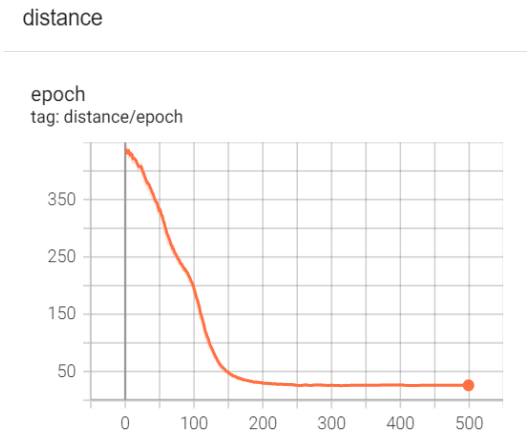
4.2 DNN Q-LEARNING 1: Q-TABLE LOSS

Q-Table 을 DNN 을 통해 학습하기 위한 손실함수는 다음과 같이 설정하였다. 신경망을 통해 예측된 Q-Table 과 기존 Q-Table 간의 Mean Squared Error(MSE)를 손실 함수로 사용하였으며, 아래의 <수식 1>은 이 과정을 도식화 한다.

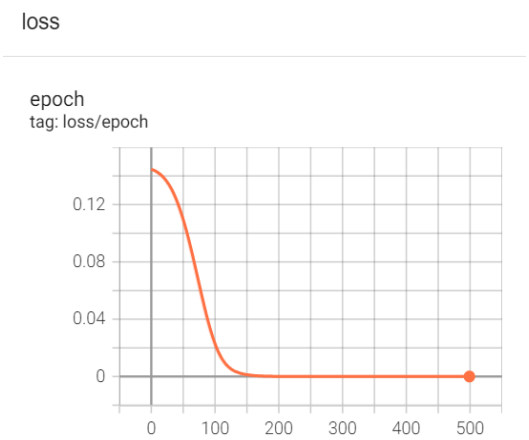
$$\text{MSE Loss} = \frac{1}{N}\sum_{i=1}^{N}(Q_{\text{target},i} - Q_{\text{predicted},i})^2$$

<수식 1> DNN Q-Learning Loss Function

이를 통해 학습한 결과 네트워크 출력 Q-Table 로 구성한 최적 Paht 의 Distance 와 학습과정에서의 Loss 변화는 <그래프 1>과 <그래프 2>와 같다.



<그래프 1> DQL 1 Distance



<그래프 2> DQL 1 Loss

하지만 이 접근 방식에는 몇 가지 중요한 문제점이 있다:

1. 예측 Q-Table 의 동일화: 학습 과정에서 신경망이 기존 Q-Table 과 동일한 값을 예측하도록 최적화되면, 더 이상 학습이 진행되지 않는다. 이는 예측된 Q-Table 이 항상 기존의 Q-Table 과 일치하도록 학습이 진행되기 때문이다.
2. 경로 출력의 변화 없음: 초기 Q-Table 을 사용하여 구성된 경로와 최종 학습 후의 경로가 동일하게 유지된다. 이는 학습이 새로운 정보를 학습하지 못하고, 단순히 기존의 정보를 복사하는 데 그친다는 것을 의미한다.

초기 Q-Table 로 구성한 경로와 학습 후의 경로는 다음과 같다:

- Initial path before training:
  [0, 862, 66, 918, 944, 922, 971, 426] ... [25, 394, 313, 7, 322, 798, 997, 988]
- Final path after training:
  [0, 862, 66, 918, 944, 922, 971, 426] ... [25, 394, 313, 7, 322, 798, 997, 988]

이와 같이 학습 전후의 경로가 동일하게 유지되는 현상은 신경망이 새로운 패턴을 학습하지 못하고, 기존의 Q-Table 을 단순히 모사하는 데 그쳤음을 보여준다.

다음 절에서는 이러한 문제를 해결하기 위한 개선된 접근 방식을 제안하고, 보다 효과적인 Q-Table 학습 방법을 탐구한다.

4.3 DNN Q-Learning 2: Q-Table with Distance Loss

이전 학습과정에서는 출력 Q-Table 이 기존 Q-Table 과 완전히 동일해지는 문제를 가지고 있었다. 이번 절에서는 기존의 Q-Table 과 예측된 Q-Table 간의 Mean Squared Error(MSE)와 경로 비용(path cost)을 가중치로 조합한 손실 함수를 활용하여 Q-Table 을 개선하는 방법을 설명한다

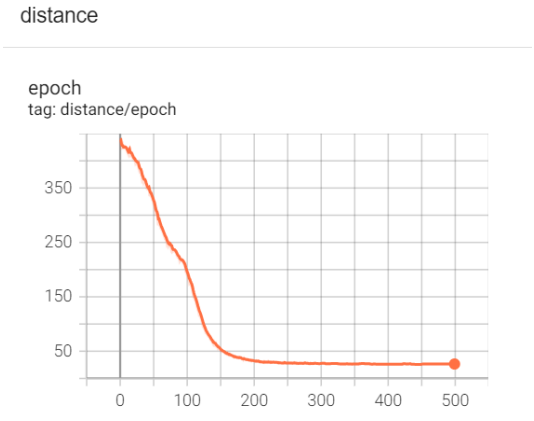
개선된 손실 함수는 다음과 같이 정의된다. 기존 Q-Table 과 네트워크를 통해 예측된 Q-Table 간의 MSE 와 예측된 Q-Table 을 통해 계산된 최적 경로의 비용을 조합하였으며, 이를 도식화한 수식은 아래의 <수식 2>와 같다.

$$\text{Loss} = \alpha \cdot \text{Loss}_{\text{MSE}} + \beta \cdot \text{Path Cost}_{\text{normalized}}$$

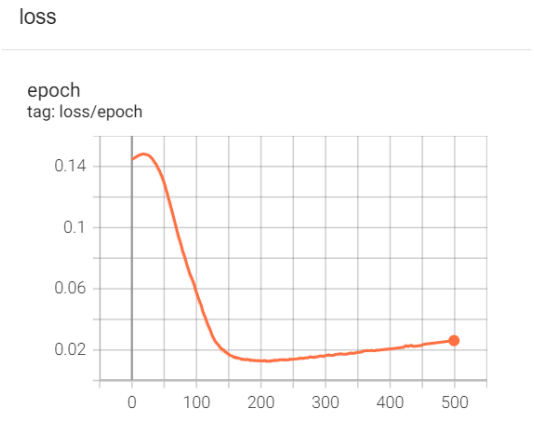
<수식 2> 개선된 Loss Function

개선된 손실 함수는 기존의 loss 와 최적 경로 비용을 조합하기 위해 경로 비용에 적절한 스케일링 작업이 이루어졌으며, 각 loss 에 가중치를 두어 학습이 진행될수록 loss 에 대한 가중치를 두었다. 학습 초기에는 MSE 손실에 더 큰 비중을 두고, 후반으로 갈수록 경로 비용 손실에 더 큰 비중을 두어 학습을 진행한다. 이는 초반에는 기존 Q-Table 과 유사한 방향으로 빠르게 수렴하고, 후반에는 거리 비용에 더 많은 영향을 받아 최적화된 경로를 찾도록 한다.

이를 통해 학습한 결과 네트워크 출력 Q-Table 로 구성한 최적 Paht 의 Distance 와 학습과정에서의 Loss 변화는 <그래프 3>과 <그래프 4>와 같다.



<그래프 3> DQL 2 Distance



<그래프 4> DQL 2 Loss

이전 Loss 와 비교해보면, 이전 Loss 는 0 에 수렴하는 모습을 보였으나 개선된 Loss 는 후반으로 갈수록 path cost 의 영향을 크게 받아 약간 증가하는 경향을 보인다.

학습 후반으로 갈수록 실제 경로를 뽑아보면 기존 Q-Table 로 구성한 경로와는 많은 부분이 변경되며, 기존보다 더 나은 경로를 찾기도 한다. 이는 아래의 경로 비교를 통해 확인할 수 있다.

- Initial path before training:
  [0, 862, 66, 918, 944, 922, 971, 426] ... [25, 394, 313, 7, 322, 798, 997, 988]
- Final path after training:
  [0, 862, 66, 918, 944, 922, 971, 426] ... [194, 108, 322, 7, 313, 997, 798, 753]

이 경로를 바탕으로 Distance 를 계산한 결과 기존 Q-Table 은 25.9, DNN 출력 Q-Table 은 25.0 으로 거리가 소폭 줄어든 것을 확인할 수 있다. 위의 결과는 신경망이 단순히 기존 Q-Table 을 모방하는 데 그치지 않고, 새로운 최적 경로를 학습하여 실제 경로의 효율성을 향상시켰음을 보여준다. 이를 통해 Q-Learning 의 성능을 더욱 효과적으로 개선할 수 있다.

5 DEEP Q-NETWORK

이번 장에서는 Deep Q-Network(DQN)를 활용하여 Direct Q-Value 추론 과정을 살펴보고, 제안된 방식을 통해 솔루션을 개선하는 과정을 보여준다. DQN 의 기본 개념과 작동 원리를 이해하고, 이를 통해 효율적인 솔루션을 개발하는 과정을 단계별로 설명한다.

5.1 제공된 DQN 모델의 학습과정 분석

제공된 DQN 모델의 기본적인 학습 과정은 다음과 같다.

1. 현재 상태에서 Q-Value 예측:

   신경망은 현재 상태(state)에서 가능한 모든 행동(action)에 대한 Q-값을 예측한다.

2. 타겟 계산:

   현재 보상(reward)과 다음 상태(next state)에서의 최대 Q-값(max Q-value)을 통해 타겟(target)을 계산한다.

   보상(reward)은 현재 상태에서 다음 상태까지의 거리의 음수 값이다.

   타겟 계산 식:은 다음 수식 3 과 같다. 이때 γ는 할인율(GAMMA)이다.

$$\text{target} = \text{reward} + \gamma \times \text{best\_q\_value}$$

<수식 3> target 계산 식

3. 손실 계산 및 네트워크 업데이트:

   현재 상태에서의 Q-값과 타겟 값을 비교하여 손실(loss)을 계산한다.

   이 손실을 기반으로 신경망을 업데이트한다.

메모리(memory)와 경험(experience)를 사용한 배치 학습(batch update) 과정은 다음과 같다.

- Experience: 현재 step 은 2 로 되어 있으므로, 한 에피소드(episode)에서 현재 state 에서 다음 action 에 대한 정보들을 포함한다.

경험(experience)의 구성 요소는 다음과 같다: 현재 상태(state), 현재 상태의 텐서(state tensor), 행동(action), 보상(reward): 현재 상태 보상과 다음 상태 보상의 합, 다음 상태(next state), 다음 상태의 텐서(next state tensor)

- Memory: 경험(experience)들을 저장해 놓는 공간이다.
- Batch: Memory 에서 네트워크 학습에 사용할 experience 단위(개수)이다.
- Batch Update: 메모리에 경험이 추가되고, 경험의 수가 배치 크기(batch size) 이상이 되면, 새로운 경험이 추가될 때마다 메모리에 있는 모든 경험을 반복하여 학습한다.

제공된 코드의 비효율성은 메모리에 경험이 추가될 때마다 발생한다. 현재는 메모리에 경험이 추가될 때, 메모리에 저장된 경험이 배치 크기를 초과하면, 새로운 경험이 추가될 때마다 메모리에 있는 모든 경험을 반복하여 학습한다. 이로 인해 학습 시간이 매우 오래 걸리게 된다.

5.2 DQN 1: 네트워크 재구성 및 최적화

이번 절에서는 제공된 DQN 네트워크와 학습과정을 최적화하는 과정을 설명한다. 네트워크 구조와 학습 과정에서의 변화를 통해 기존 코드의 비효율성을 개선하고, 학습 속도를 향상시킨다. 우선 새롭게 구성된 DQN 네트워크는 4 개의 레이어로 이루어져 있으며, 각 레이어는 다음과 같은 역할을 한다.

- 입력층: 현재 상태(state)의 정보를 입력으로 받는다.
  - 현재 위치: 998 개의 가능한 위치 중 현재 위치를 인덱스로 표시하며, 해당 인덱스는 1 로, 나머지는 0 으로 표시된다.
  - 방문 도시: 에이전트가 방문한 도시는 해당 인덱스에 1 로 표시된다.
- 출력층: 현재 상태에서의 Q-값(Q-value)을 출력한다.
- 마지막 레이어의 입력과 출력은 Q-값과 동일한 차원으로 맞춰진다 (5.4 DQN 3: 가중치 초기화 절에서 자세히 다룸).

학습 과정은 기존 코드와 유사한 방식으로 진행되지만, 메모리 관리와 배치 업데이트에서 몇 가지 중요한 변경 사항이 있다

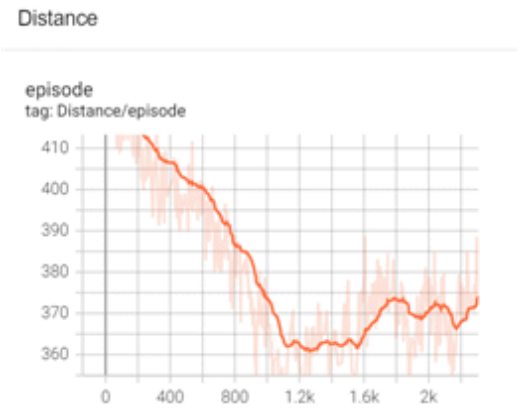
기본 학습 원리는 다음과 같다.

- Q-Value 예측: 현재 상태에서 Q-값을 예측.
- 타겟 계산: 현재 보상(reward)과 다음 상태에서의 최대 Q-값(max Q-value)을 통해 타겟을 계산.
- 손실 계산: 예측된 Q-값과 타겟 값 간의 차이를 손실 함수로 계산.
- 네트워크 업데이트: 손실을 기반으로 신경망을 업데이트.

메모리와 배치 업데이트하는 과정에서 변경 사항이 존재하며 세부적인 내용은 다음과 같다.

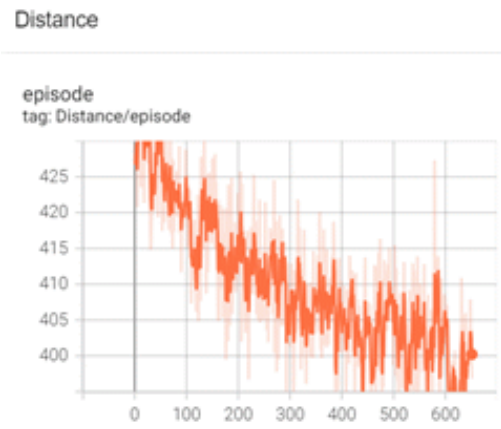
- 기존 방식: 메모리에 경험이 추가되고, 배치 크기(batch size)를 초과하면 새로운 경험이 추가될 때마다 메모리에 있는 모든 경험을 반복하여 학습한다.
- 개선된 방식: 현재 에피소드에서의 모든 경험을 메모리에 추가한 후, 한 번에 배치 업데이트를 진행한다. 이를 통해 학습 시간이 단축되고, 효율성이 향상된다.
- 실제 하나의 에피소드를 학습하는 데에 소요되는 시간은 90s 에서 2s 로 감소하였다.

아래 그래프는 개선된 모델과 학습 방법으로 학습한 결과이다. <그래프 5>와 <그래프 6>은 2400 에피소드 학습한 그래프로 1100 에피소드 이후 수렴한 모습을 보이며 추가로 에피소드를 실행할수록 발산하는 형태를 보인다. <그래프 7>과 <그래프 8>은 600 에피소드 학습한 결과이며, 2400 에피소드 학습한 결과 loss 가 충분히 감사하여 튀는 것처럼 보이는 부분에 대해 학습이 정상적으로 진행됨을 보이기 위한 그래프이다.

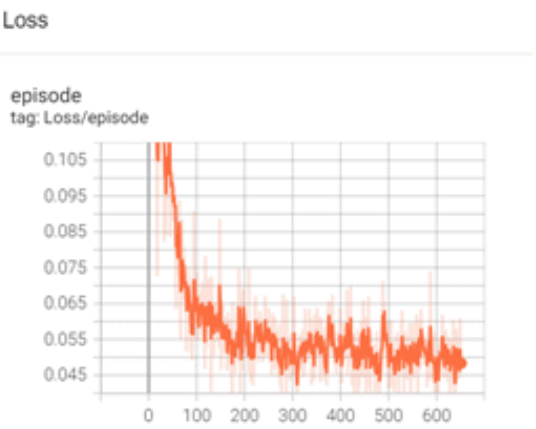


<그래프 5> DQN 1 Distance

<그래프 6> DQN 1 Loss



<그래프 7> DQN 1 Distance



<그래프 8> DQN 1 Loss

2400 에피소드 학습 결과 minimum distance 는 337 을 찍었으며, 수렴 이후 평균적으로 440 에서 360~370 수준으로 떨어진 것을 확인할 수 있다.

5.3 DQN 2: Q-TABLE 활용

이번 절에서는 Q-Table 을 활용하여 DQN 모델의 손실 함수를 개선하는 방법을 설명한다. 이를 통해 DQN 의 학습 성능을 향상시키고, 더 나은 행동 예측을 가능하게 한다.

DQN 2 모델의 기본적인 네트워크 구조는 DQN 1 모델과 동일하게 구성된다.

- 입력: 현재 상태(state)의 정보 (현재 위치와 방문 도시)
- 출력: 현재 상태에서의 Q-값(Q-value)

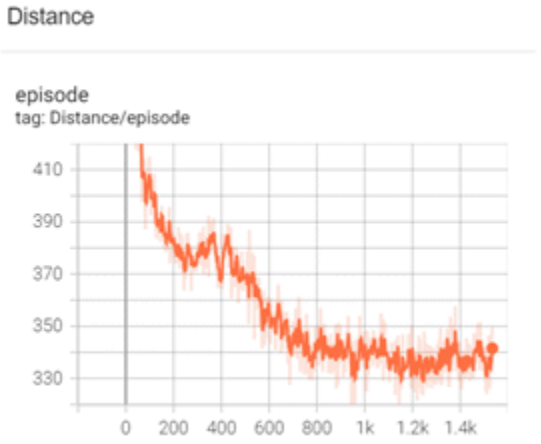
DQN 2 에서는 DQN1 의 기본 손실 함수에 Q-Table 을 활용하여 추가적인 손실 항목을 더한다. 구체적으로, 현재 Q-값과 Q-Table 에서 현재 행동(action)에 대한 Q-값 간의 MSE 를 추가한다. 최종 손실 함수는 <수식 4>와 같이 정의된다

$$\text{loss} = \text{loss}_{\text{return}} + \text{loss}_{\text{q\_table}}$$

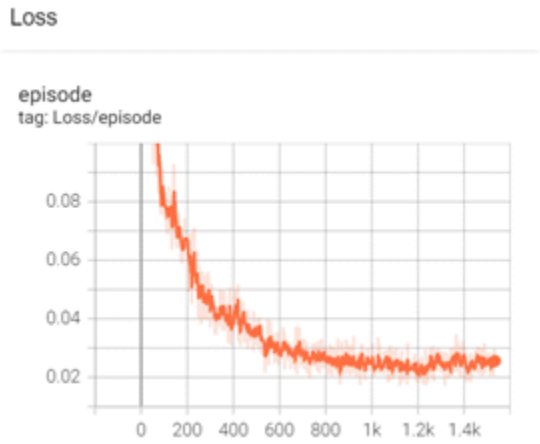
<수식 4> DQN 2 Loss

DQN 2 의 개선된 손실 함수는 신경망이 기존 Q-Table 의 정보를 활용하여 더 정확한 Q-값을 학습하도록 유도한다. 이를 통해 학습 초기에는 Q-Table 을 통해 빠르게 학습하고, 이후에는 신경망 자체의 예측 성능을 향상시킬 수 있다.

아래 <그래프 9>와 <그래프 10>은 1400 에피소드 학습한 결과로 minimum distance 는 315 를 찍었으며 수렴 이후 평균적으로 440 에서 330~340 수준으로 떨어진 것을 확인할 수 있다. 이를 통해, DQN 1 모델과 비교하여 약 10% 정도의 성능이 향상된 것을 확인할 수 있다.



<그래프 9> DQN 2 Distance



<그래프 10> DQN 2 Loss

5.4 DQN 3: 가중치 초기화

이번 절에서는 DQN 모델의 가중치 초기화 방법을 다룬다. 네트워크의 마지막 레이어 가중치를 Q-table 과 동일한 차원으로 설정하고, 이를 거리 행렬(distance matrix)로 초기화하여 초기 학습 성능을 향상시키는 방법을 다룬다.

DQN 3 모델의 기본적인 네트워크 구조는 DQN 1 모델과 동일하게 구성된다.

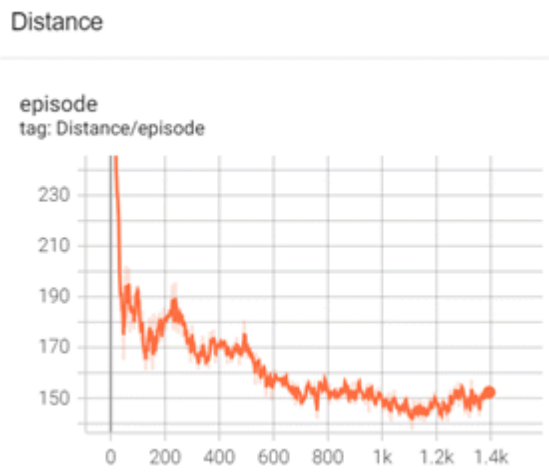
- 입력: 현재 상태(state)의 정보 (현재 위치와 방문 도시)
- 출력: 현재 상태에서의 Q-값(Q-value)

네트워크의 마지막 레이어는 앞서 "5.2 DQN 1: 네트워크 재구성 및 최적화"에서 언급된 바와 같이 입력과 출력 차원이 Q-Value 와 동일하게 맞추어져 있는데, 여기에 Q-Value 의 차원은 도시 정점의 개수인 998 로 설정된다. 따라서, 마지막 레이어의 가중치 차원은 998×998 이 되며, 이 가중치를 초기화할 때 거리 행렬을 사용하였다. 거리 행렬은 도시 간의 거리를 나타내며, 이를 가중치로 설정하면 네트워크가 초기 학습 단계에서 가까운 도시를 선택할 확률이 높아진다. 이 방식은 네트워크가 가장 좋은 Q-값을 선택할 때 가까운 도시를 선택할

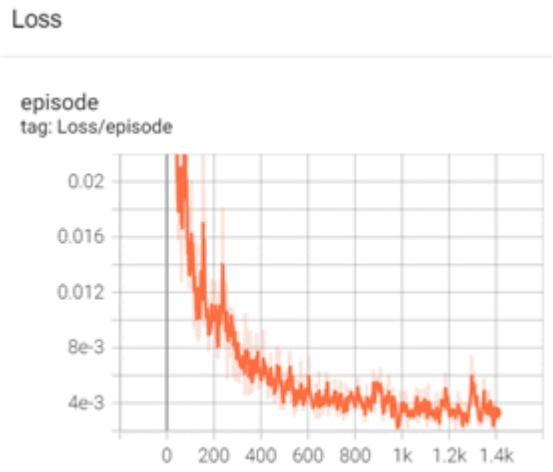
가능성을 높여준다. 비록 이 방식이 최적의 경로를 보장하지는 않지만, 초기 학습 단계에서 탐욕적(greedy)인 탐색을 통해 대부분의 경우 합리적인 판단을 내릴 수 있다.

DQN 3 의 손실 함수는 기본 DQN 1 의 손실 함수와 동일하다.

아래 <그래프 10>와 <그래프 11>은 1400 에피소드 학습한 결과로 minimum distance 는 137 을 찍었으며 수렴 이후 평균적으로 320 에서 150 수준으로 떨어진 것을 확인할 수 있다. 이를 통해, DQN 1 모델과 비교하여 약 60% 정도의 성능이 향상된 것을 확인할 수 있다.



<그래프 11> DQN 3 Distance



<그래프 12> DQN 3 Loss

5.5 DQN 4: Q-TALBE 활용 + 가중치 초기화

이번 절에서는 DQN 모델의 Q-Table 활용과 가중치 초기화 방법을 다룬다. DQN 4 모델은 기본 loss 에 Q-Table 의 loss 를 추가한 DQN 2 모델과, 네트워크의 마지막 레이어 가중치를 거리 행렬(distance matrix)로 초기화하는 DQN 3 모델을 융합한 모델이다..

DQN 4 모델의 기본적인 네트워크 구조는 DQN 1 모델과 동일하게 구성된다.

- 입력: 현재 상태(state)의 정보 (현재 위치와 방문 도시)
- 출력: 현재 상태에서의 Q-값(Q-value)

DQN 4 모델은 학습 초기 단계에서 압도적으로 빠른 수렴을 보였다. 약 50 에피소드 만에 학습이 완료되었으며, 더 이상의 학습 진행이 이루어지지 않았다. 이는 DQN 2 와 DQN 3 모델 각각의 특징을 결합하여 초기 학습 속도를 극대화한 결과로 볼 수 있다.

그러나, 기대와는 달리 DQN 4 모델의 성능은 DQN 3 모델보다 떨어졌다. 이는 초기 단계에서 네트워크에 너무 많은 정보를 제공한 결과로 분석된다. 초기 학습 단계에서 과도한 정보는 네트워크가 지나치게 빠르게 수렴하게 하여, 탐색을 충분히 하지 못하고, 더 나은 정책을 학습할 기회를 놓친 것으로 분석된다.

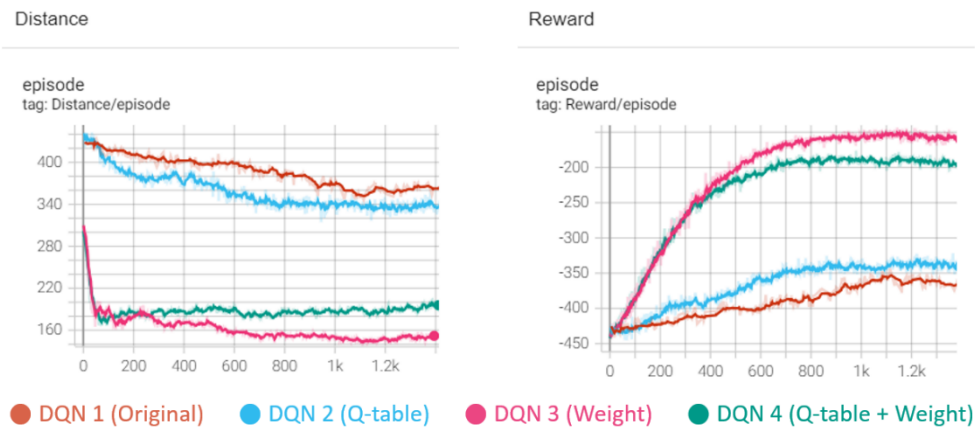
아래 <그래프 11>와 <그래프 12>은 200 에피소드 학습한 결과로 수렴 이후 평균적으로 320 에서 180 수준으로 떨어진 것을 확인할 수 있다. 이를 통해, DQN 1 모델과 비교하여 약 50% 정도의 성능이 향상된 것을 확인할 수 있다. 학습 그래프에서 확인할 수 있다시피 이전 DQN 모델들과 비교하여 굉장히 빠른 수렴 속도를 보인다.

<그래프 11> DQN 4 Distance

<그래프 12> DQN 4 Loss

5.6 DQN 모델 별 성능 비교

이번 절에서는 DQN 모델별로 학습 성능을 비교하고, 각 모델의 distance 와 reward 측면에서의 차이를 분석한다(그래프 13). 네 가지 DQN 모델(DQN 1, DQN 2, DQN 3, DQN 4)을 비교하여 가중치 초기화와 Q-Table 활용의 영향을 평가한다.



<그래프 13>

아래는 DQN 1 과 DQN 2 의 비교이다.

- 가중치 초기화가 없는 DQN 1 과 DQN 2 모델은 가중치 초기화가 된 DQN 3 와 DQN 4 에 비해 현저히 떨어지는 성능을 보였다.
- DQN 2 는 DQN 1 과 비교했을 때, 손실 함수에 Q-Table 정보를 추가하여 성능이 개선되었다. 이는 distance 와 reward 측정을 통해 확인할 수 있었다.

다음으로 DQN 3 과 DQN 4 의 비교이다.

- DQN 4 모델은 초기에 훨씬 빠르게 수렴함을 확인할 수 있었다. 단 50 에피소드 만에 학습이 완료되는 현상을 보였다.
- 그러나, 학습 후반부로 갈수록 DQN 3 모델은 distance 가 점진적으로 감소한 반면, DQN 4 모델은 오히려 distance 가 증가하는 경향을 보였다.

가중치 초기화가 적용된 DQN 3 와 DQN 4 모델은 DQN 1 과 DQN 2 에 비해 초기 학습 성능이 뛰어났다. 특히, DQN 4 는 초기에 매우 빠른 수렴을 보였다. 그러나, 초기 단계에서 과도한 정보 제공은 장기적으로 학습 성능을 저하시킬 수 있다. 이는 DQN 4 모델의 distance 가 학습 후반부에 증가한 이유로 설명된다.

DQN 을 사용했을 때 거리가 20 대 이하로 떨어지지 않은 이유는 기본적인 DQN 의 학습 원리와 관련이 있을 것으로 판단된다. DQN 모델은 작은 데이터셋에서는 크게 효율적이지 않지만, 데이터가 많은 상황에서의 확장이 가능하다. 현재 1000 개 정점의 TSP 문제는 A* 알고리즘이나 다른 Value-Based 방법으로도 좋은 결과를 얻을 수 있으므로, DQN 이 오히려 비효율적일 수 있다. 그럼에도, 아무런 정보 없이 돌린 모델과 Q-Table 또는 가중치 초기화를 거친 후 학습한 모델 간의 성능 향상은 명확히 존재했으며, 이는 학습 과정의 정당성에 문제가 없음을 보여준다.

6 결론

본 프로젝트에서는 약 1000 개의 도시를 포함하는 복잡한 Traveling Salesman Problem 을 해결하기 위해 여러 가지 방법의 강화 학습을 사용해 보았다. 이러한 접근 방식은 TSP 와 같은 복잡한 최적화 문제에서 전통적인 알고리즘의 한계를 극복하고, 높은 탐색 효율성과 적응력을 바탕으로 유연한 솔루션을 제공할 수 있는 잠재력을 지니고 있다.

Value-based Policy Extraction 에선 초기 학습 시간을 줄이기 위해 A* 알고리즘을 사용해 최적 경로를 구한 후, 이를 기반으로 초기 value 함수를 설정하였다. 이러한 초기화 작업을 통해 가치 반복 작업을 수행하였고, 최적의 value table 을 구성할 수 있었다. 이 과정은 전체적인 학습 효율성을 크게 향상시켰다. 그럼에도 초기 경로보다 좋지 않은 결과가 나온 것은 초기 경로가 이미 최적이거나 초기 경로가 지역 최적해에 빠져서 발생한 것으로 고려할 수 있다. 이후 연구를 통해 더 다양한 초기 경로 설정을 사용해 보는 것이 좋아 보인다.

다음으로, Q-Learning 기법을 활용하여 몬테 카를로(Monte Carlo, MC)와 시간차(Temporal Difference, TD) 학습을 수행하였다. 초기 Q-테이블은 앞서 Value Iteration 으로 구한 V 값을 기반으로 구성하였다. MC 기법은 전체 에피소드를 통해 Q-값을 업데이트하지만, 별로 좋은 결과를 보이지 않았다. 반면에, TD 기법은 현재 상태에서 다음 상태로의 전이만을 고려하여 Q-값을 업데이트하였고, 괜찮은 결과를 만들어냈다. 이는 TD 기법이 TSP 문제에 더 적합함을 시사한다.

마지막으로, Deep Q-Network(DQN)을 활용하여 value table 학습을 고도화하고, 직접적인 Q-value 추론을 수행하였다. DQN 은 Q-테이블을 사용하지 않고 신경망을 통해 상태-행동 쌍의 가치를 근사함으로써 학습 효율성과 적응력을 크게 향상시킬 수 있는 가능성을 보였으나, 여러 모델을 사용해도 A* 알고리즘으로 구한 최적 경로보다 나은 결과에 도달하지 못했다. 이는 DQN 이 TSP 문제에 대해 추가적인 최적화와 조정이 필요함을 나타낸다. 그럼에도, 아무런 정보 없이 돌린 모델과 Q_Table 또는 가중치 초기화를 거친 후 학습한 모델 간의 성능 향상은 명확히 존재했으며, 이는 학습 과정의 정당성에 문제가 없음을 보여준다. 또한 이번 프로젝트를 통해 다양한 신경망 구조와 학습 전략을 시도해 봄으로써, 강화 학습의 실질적인 응용 가능성을 평가하는 중요한 경험을 얻었다.

결론적으로, 본 프로젝트는 강화 학습 기법을 활용한 TSP 문제 해결의 가능성을 보여주었으며, 향후 다양한 최적화 문제에 이 기법을 적용하여 더 나은 성과를 도출할 수 있는 기반을 마련하였다. 이는 복잡한 문제 해결에 있어 강화 학습의 유용성을 입증하는 중요한 사례가 될 것이다. 앞으로의 연구에서는 초기 경로 설정의 다양화, TD 기법의 최적화, 그리고 DQN 의 성능 향상을 위한 추가적인 전략을 탐구할 필요가 있다.