자율주행차량 환경을 위한
심층강화학습 기법 비교

Comparing Deep Reinforcement Learning Algorithms in Autonomous Driving Environment

Jin Uk Cho, brbl@g.skku.edu

[1] 연구과제의 개요

1. 연구 배경 및 목적

강화학습은 인공지능 분야에서 제어, 최적화, 스케줄링과 같은 일을 수행할 수 있는 매우 핵심적인 기법이다. 그 중 시스템제어의 일환으로, 자율주행자동차 제어 및 응용 분야에 대한 활발한 연구가 진행되고 있다. 한 예로, 2018 년 11 월 아마존은 AWS Deep racer 이라는 3D 경주 시뮬레이터를 배포하였다. 이는 직접 강화학습 모델을 사용해 아마존에서 제공하는 가상환경 경주 시뮬레이터에서 트랙을 학습하고 테스트하는 과정의 반복을 진행할 수 있다. 그리고 현실의 1/18 크기의 실제 자동차를 구매하여 실제 환경에서 자율주행차량을 실험해볼 수 있기도 하다. 이와 같이 가상의 세계 강화학습은 기존의 기계학습방법과 많이 다른 접근 방식으로 초기 데이터 없이 단기적인 의사결정을 내리며 장기적인 목표를 위한 행동을 취할 수 있다는 면에서 현재 많은 관심을 가지고 있다. 이번 연구에서는 강화학습 중 심층강화학습(Deep Reinforcement Learning, DRL)알고리즘들을 다양한 환경에 적용해보아, 시스템 제어에 효과적인 알고리즘들을 연구하고자 한다.

1.1. 연구 범위

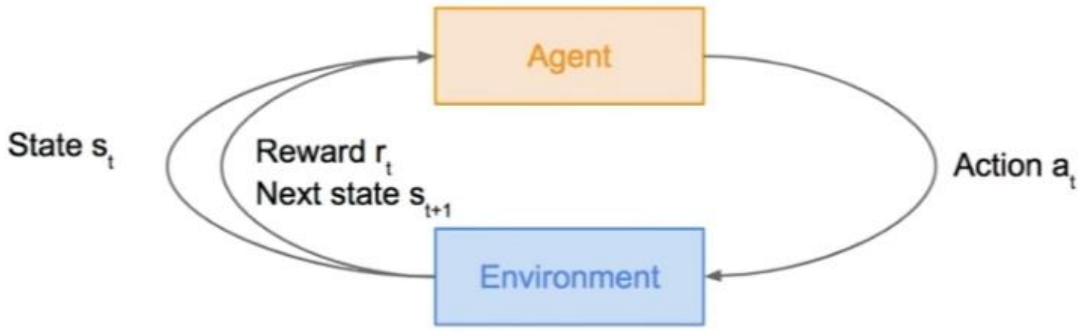
심층강화학습의 연구는 크게 Value Function Approximation 방법들과 Policy Gradient 방법 두 갈래로 나뉘어 있다. 둘의 가장 큰 차이는 강화학습의 주체(에이전트)가 할 수 있는 행동의 분포이다. Value Function Approximation 의 경우 불연속적인 행동을 하는 경우에 적합하며, 연속적인 행동을 하는 상황에선 대부분 Policy Gradient 기법들을 사용한다.

자율주행차량의 경우 모터의 각도, 가속의 정도, 브레이크의 정도 등 연속적인 행동을 위주로 제어가 이루어질 것이다. 그렇기 때문에 우리는 연속적인 행동에 맞는 Policy Gradient 기법을 위주로 탐색해볼 것이며, Value Function Approximation 기법 또한 비교의 대상으로 실험해보고자 한다. OpenAI gym 에서 제공하는 다양한 시스템제어 환경에서 그 기법들을 통해 학습시키면서 각 기법들이 유효함을 일반화할 것이다.

1.2. 배경 이론

1.2.1. Reinforcement Learning

강화학습은 주어진 환경(Environment)과 상호작용하면서 누적된 보상(Reward)을 최대화하는 것을 목표로 한다. 강화학습에 어떠한 목표가 주어지며, 관측을 통해 매번 현재 상태(state)를 알려준다. 환경은 주어진 목표에 맞는 보상체계를 가지고 주체(Agent)가 어떠한 행동(Action)을 취할 때마다 그 행동이후 바뀐 새로운 상태와 행동에서 얻어진 보상을 주체에게 돌려준다.



[그림 1] Action, Environment, State 그리고 Reward 에 대한 설명

이러한 방식을 이용하면 레이블이 지정된 훈련데이터가 필요 없어지며, 매우 복잡한 동작을 수행해야 하는 문제에서 지도학습(Supervised Learning)을 대체할 수 있다. 예를 들어, 운전하는 방법을 학습해야 한다면, 지도학습은 각 상황에 맞는 최적의 행동들에 대한 모든 조합의 데이터셋을 가지고 있어야한다. 복잡한 상황일 수록 이러한 방법은 거의 불가능에 가까울 것이다. 이와는 달리 강화학습은 주어진 데이터가 없는 상태에서 환경과의 상호작용만을 통해 학습을 해 나간다는 측면에서 복잡한 상황을 학습하는 분야에서 매우 유용하다고 할 수 있다.

1.2.2. Deep Reinforcement Learning

Dynamic Programming 과 그 이후 개발되어져 온 기존의 강화학습 알고리즘들은 Markov Decision Process 라고 정의되어질 수 있는 환경에선 이론적으로 가치함수(Value Function)를 모든 상태와 행동에 대해서 찾아낸다는 수렴성을 보장할 수 있었다. 그러나 학습에 넣을 입력 값인 상태의 크기가 커지고 복잡해지면서, 계산 복잡도와 차원의 저주라는 문제가 생긴다.

그렇기 때문에, 에이전트가 가치함수를 온전히 알아내려 하지 않고 대신 추정(Function Approximation)해내는 새로운 접근법이 생긴다. 그리고 최근 Deep Learning 연구가 매우 활발히 진행되면서, 다른 Function Approximator 의 한계를 극복해내는 Deep Neural Network 를 이용해서 값을 추정해내기 시작한다. 우리가 심층강화학습(Deep Reinforcement Learning, DRL)이라고 말하는 이 기법들의 가장 유명한 예시로는 2016 년도 이세돌과의 경기에서 놀라운 성능을 보였던 AlphaGo 가 있다. 이를 만들 때 사용한 알고리즘은 Deep Q Network, DQN 이었다.

1.2.3. Policy Gradients

기존의 가치함수를 기반으로 행동을 하고 가치함수를 업데이트 하면서 학습을 하는 방식의 강화학습 에는, 가치함수의 작은 변화에도 정책이 크게 변할 수 있다는 단점이 존재했다. 이에 대한 새로운 접근 방식으로, 정책 자체의 점진적인 변화를 통해 최적의 정책을 찾아가는 Policy Gradient 기법이 개발되었다.

Policy Gradient 기법은 확률론적(stochastic)인 방법과 결정론적(deterministic)인 방법으로 나뉘며, 확률론 적인 방법은 정책을 하나의 행동 확률 분포로 가정한다. 결정론적인 방법은 Value Function Approximation 과 비슷하게 어떤 정책은 하나의 상태에 하는 행동이 정해져 있다. 확률론적인 방법으로 유명한 알고리즘은 A2C,(Advatage Actor Critic), TRPO(True Region Policy Optimization), PPO(Proximal Policy Optimization) 가 있으며, 결정론적인 방법으로 유명한 알고리즘은 DDPG(Deep Deterministic Policy Gradient) 이다.

A. Deep Deterministic Policy Gradient, DDPG

기존의 Value Function Approximation 기법은 이산적인 행동을 하는 경우 우수한 성능을 보여주었다. 그러나, 연속적인 행동을 하는 차원에서는 매 timestep 마다 연속적인 행동을 모두 계산하기에 계산자원의 한계 및 차원의 저주 문제가 발생한다. 이러한 방식에 대한 다른 접근 방법으로, 불연속 적인 제어 문제를 해결하기 위해 Policy Gradient 대안으로 제시되었으며, 그 중에서도 off-policy, actor-critic 알고리즘인 심화 결정론적 정책경사 DDPG(Deep Deterministic Policy Gradient) 알고리즘이 개발되었다.

원래 상태 s_t에서 행동 a_t를 할 때 받기를 기대하는 보상 Q는 다음과 같다.

$$Q^{\pi}(s_t, a_t) = E_{r_t, s_t \sim E}[r(s_t, a_t) + \gamma E_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]]$$

DDPG 에선 정책이 결정론 적이기 때문에, 각 상태에서 할 행동이 확률적이지 않고 정해져 있다.

$$Q^{\pi}(s_t, a_t) = E_{r_t, s_t \sim E}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))]$$

이는 각 행동에 대한 기대되는 값이 더 이상 정책에 의존적이지 않게 되기 때문에, off policy 를 사용할 수 있다는 의미이다. DDPG 는 기존 Policy Gradient 의 유명한 기법인 actor critic 에, off policy 를 적용하기 위해 DQN 에서 사용되었던 Replay Buffer 와 Target Q network 를 사용한다. 그리고, Deep Neural Network 를 통해 점진적으로 policy 를 업데이트 하는 방식을 취한다. DDPG 의 목적함수 식은 다음과 같다.

$$\max_{\theta} J(\mu_\theta) = E[r_1^{\gamma} | \mu]$$

$$= \int_S \rho^{\mu}(s) r(s, \mu_\theta(s)) ds = E_{s \sim \rho^{\mu}}[r(s, \mu_\theta(s))]$$

$$\nabla_{\theta^\mu} J \approx \sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

위 수식에서 μ_θ 는 정책의 표현하는 신경망의 가중치 θ 에 의해 결정된 어떤 정책을 의미한다. 결국 목적함수식에선 이에 대해 샘플링된 Policy Gradient 를 가지고 목적함수가 최대가 되도록 업데이트를 진행한다. 또 정책을 평가하기 위해, 손실함수를 정의하여 손실함수의 값이 최소가 되도록 θ를 조정한다.

$$\mathrm{L} = \frac{1}{\mathrm{N}} \sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$$

B. Proximal Policy Optimization, PPO

Policy Gradient 알고리즘으로서 최근 가장 좋은 성능을 보이고 있는 것은 PPO, 근위 정책 최적화 알고리즘이다. PPO 알고리즘은 Trust Region Policy Optimization, TRPO 에 뿌리를 두고 있다. TRPO 에서의 목적함수 수식은 다음과 같다.

$$\max_\theta \hat{\mathrm{E}}_\mathrm{t}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}\hat{A}_t\right]$$

$$\text{subject to } \hat{\mathrm{E}}_\mathrm{t}\left[\beta KL[\pi_{\theta old}(\cdot|s_t), \pi_\theta(\cdot|s_t)]\right] \leq \delta$$

위 수식에서 $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}$ 는 현재 정책과 새로운 정책 간의 Probability Ratio 이다. 이 비율은 새로운 정책이 어떤 방향으로 탐색할지를 알려준다. 여기에 $\hat{A}_t$ 는 새로운 정책으로 샘플링한 각각의 행동이 좋은 행동 이였는지를 판별하는 Advantage Function 의 추정치이다.

또, TRPO 의 가장 큰 특징은 다른 Policy Gradient 알고리즘에서 한계로 꼽히는 매개변수의 미세한 변화에 따른 높은 Variance 를 막기 위해, 매개변수의 업데이트에 상한을 두는 제약을 부과한다는 것이다. 여기서 쓰이는 제약은 $\beta KL[\pi_{\theta old}(\cdot|s_t), \pi_\theta(\cdot|s_t)]$ 이며, β 는 계수이며 뒤의 수식은 쿨벡-레이블러 발산 (Kullback-Leibler divergence, KL divergence) 이다. KL divergence는 두 확률변수 간의

분포의 차이를 구해주는 일종의 거리 함수이다. 위와 같은 식으로 목적함수의 최대화를 제약함으로써 TRPO는 정책의 업데이트에 매우 큰 변화를 제한한다.

PPO 는 이와는 약간 다른 방식을 취하는데, KL divergence 를 이용했던 TRPO 와는 달리 PPO 에선 clipping 기법과 adaptive KL penalty coefficient 를 이용해 좀 더 계산 효율적이며 TRPO 의 장점인 안정성을 가져왔다고 할 수 있다.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}, r_t(\theta_{old}) = 1$$

$$\mathrm{L}^{\mathrm{CLIP}}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t]$$

위 식에서, $, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ 은 clipping 기법으로서, Advantage Function 이 양수여서 그 행동을 할 확률이 증가하든, 음수여서 그 행동을 할 확률이 감소하든 Hyperparameter 값이 ϵ 을 넘지 않도록 한다. 또, min 을 취함으로서 $r_t(\theta)\hat{A}_t$에 대한 하한을 제시해준다.

최종적으로 나오는 오류 함수는 다음과 같다.

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right], \quad (9)$$

where c_1, c_2 are coefficients, and S denotes an entropy bonus, and L_t^{VF} is a squared-error loss $(V_\theta(s_t) - V_t^{\mathrm{targ}})^2$.

$\mathrm{L}^{\mathrm{CLIP}}$ 은 Clipped 된 오류함수, L^{VF} 는 Value Function 에 대한 오류함수, $\mathrm{S}[\pi_\theta]$ 는 각 행동의 확률이 어느정도 균형있게 유지되기 위한 Entropy 값이다.

[2] 실험 방법 및 설계

1. 실험 개요

자율 주행 차량 시스템은 소프트웨어적으로나 하드웨어적으로나 많은 복잡한 기술들을 포함하고 있다. 우리가 자율주행 알고리즘을 검증하기 위해 실제 세계에서 실험을 하고자 하는 것은 돈과 시간, 노동력의 문제로 실현 가능하지 않을 것이다. 위에서 언급한 Amazon Deep racer 와 같이, 실제 운전원리를 충분히 반영한 시뮬레이션 환경에서의 알고리즘 테스트는 훌륭한 대안이 된다. 다양한 오픈소스 시뮬레이터들이 있지만, 이번 실험에서 목표로 하는 환경은 CarRacing-v0 이다. CarRacing-v0 는 Open AI gym 의 box2d 모듈 환경 중 하나로서, 2d 환경에서의 자동차 경주 시뮬레이터이다. 이번 연구의 목표는 다양한 알고리즘들의 자율주행차량에 맞는 환경에 대한 적용가능성을 탐구하는 것이므로, 시간과 자원의 한계를 고려하여 먼저 CarRacing-v0 환경에서 알고리즘 들을 적용해보기로 한다.

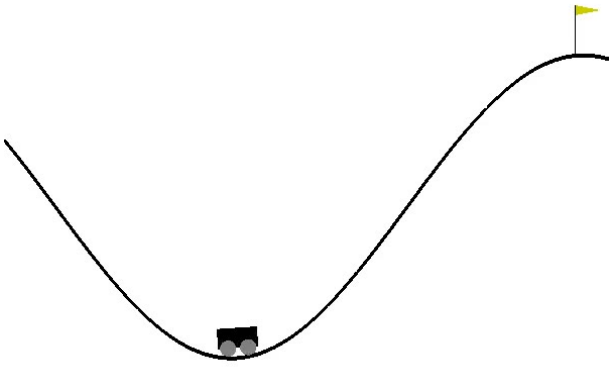
또, 실험을 위해선 먼저 Policy Gradient 알고리즘들을 가능한 다양한 환경에 적용해보고, 자율주행차량 환경에서의 적용가능성을 탐구한다. 기존 Value Function Approximation 기법과의 비교를 위해, 대표 알고리즘인 DQN 또한 적용해볼 것이다..

Algorithm	Model	Policy	Action Space	Observation Space	Operator
Q-Learning	Model-Free	Off-policy	Discrete	Discrete	Q-value
SARSA	Model-Free	On-policy	Discrete	Discrete	Q-value
DQN	Model-Free	Off-policy	Discrete	Continous	Q-value
DDPG	Model-Free	Off-policy	Continous	Continous	Q-value
TRPO	Model-Free	Off-policy	Continous	Continous	Advantage
PPO	Model-Free	Off-policy	Continous	Continous	Advantage

1.1. 실험 환경

총 3 개의 가상환경에서 실험이 이루어졌으며, 각 환경에 대한 설명은 다음과 같다.

A. MountainCar-v0



OpenAI gym에서 제공하는 기본 환경 중 하나인 MountainCar-v0는 1차원의 트랙에 있는 차를 최소한의 시도로 오른쪽 언덕에 위치시키는 것이 목표이다. 차는 처음에 두 언덕의 사이에서 시작하며, 엔진이 위로 바로 올라가기에 충분히 강력하지 않기 때문에 왼쪽과 오른쪽으로 움직이면서 관성을 이용해야 오른쪽 언덕에 도달할 수 있다.

Discrete한 환경과 연속한 환경의 보상공식은 다음과 같다

In discrete action space

$$\mathrm{R}_{\mathrm{timestep}} = timestep * -1 \qquad (timestep \leq 200)$$

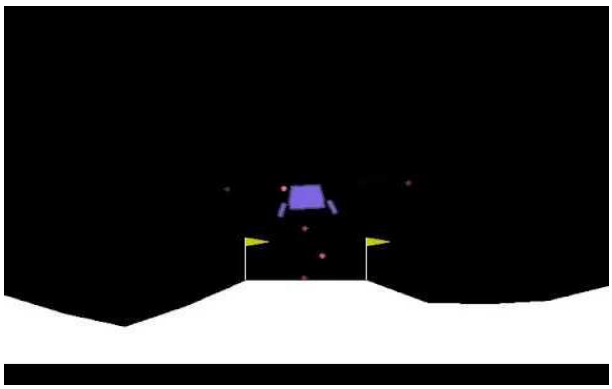
In continuous action space

$$\mathrm{R}_{\mathrm{timestep}} = 0 - \sum_{t=1}^{timestep} (action\ value\ at\ t)^2 * 0.1 \qquad (timestep \leq 999)$$

when is done, get reward $+ 100$

https://gym.openai.com/envs/MountainCar-v0/

B. LunarLander-v2

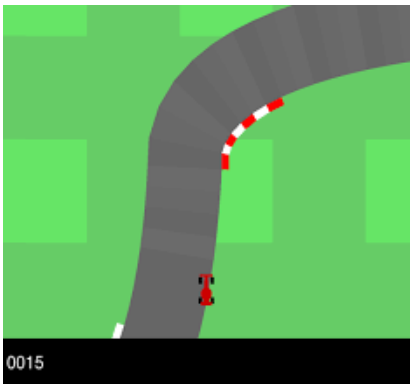


LunarLander-v2 는 2 차원 공간에서 우주선이 지면에 착륙하는 걸 시뮬레이션한 환경이다. 엔진은 시작부터 끝까지 항상 켜 있는 상태이며, 적당한 스피드를 유지하며 최소한의 시간, 즉 최소한의 연료로 두 다리가 모두 도착지점 인 깃발 사이에 안정적으로 착지하는 것을 목표로 한다.

목표도착지점은 항상 가운데 (0, 0) 지점이며, 도착지점에 정확히, 0 의 스피드로 멈춘다면 약 100.~140 의 보상을 받으나, 그 외 지점에 도착할 시 보상이 줄어든다. 시간을 줄이기 위해 frame 이 지나갈 때마다 -0.3 의 보상을 받으며, 깃발 내 지점에 도착하거나 그러지 않을 시, -100 또는 + 100 의 보상을 추가로 받는다. 마지막으로, 두 다리가 제대로 착지할 시, 다리 당 + 10 의 보상을 받을 수 있다.

https://gym.openai.com/envs/LunarLander-v2/

C. CarRacing-v0



CarRacing-v0 는 2 차원 차량주행 시뮬레이션 환경이다. 에이전트는 약 300 개 정도의 타일로 생겨난 2 차원 트랙에서, 차의 steering, gas, brake 3 가지 행동을 제어하여 트랙을 벗어나지 않으며 가능한 빠르게 움직이여야 한다. 목표는 일정수준의 보상점수를 계속 유지하는 것이다.

하나의 에피소드는 트랙을 완주하거나, 일정 시간 동안 보상을 충분히 받지 못할 시, 또는 트랙에서 벗어나 생성된 공간의 끝에 도달했을 시 종료된다. 보상은 frame 당 -0.1 을 가지며 만약 타일을 방문할 시 각 타일 당 1000/(총 타일 개수)를 받는다. 결국, 타일의 개수와 상관없이 트랙을 완주했을 경우 1000 - 0.1 * (총 걸린 frame)을 보상으로 가져간다.

https://gym.openai.com/envs/CarRacing-v0/

1.2. 모델의 구성

1.2.1 Stable Baselines

OpenAI Baselines 는 OpenAI 에서 Reinforcement Learning(RL) 알고리즘들을 실제로 구현해 바로 환경에 적용하게끔 만들어둔 응용 라이브러리이다. Stable Baselines 는 OpenAI Baselines 에서 구현된 모델들이 가졌던 오류나 문제를 보완하고 수정하여 만들어졌다.

1.2.2. Hyperparameter 설정

하이퍼 파라미터는 학습이 시작되기 전에 설정해 두어야 하는 값을 말하며, 대부분의 머신러닝 모델에서는 하이퍼 파라미터가 존재한다. 이는 데이터를 통해 추정할 수 없으며, 연구자가 직접 값을 설정해 두어야 한다. 하이퍼 파라미터는 대부분의 경우 모델의 학습 후 성능에 큰 영향을 끼치기 때문에, 그 값의 조정과 튜닝은 매우 중요하다.

이번 연구에선 하이퍼 파라미터에 큰 비중을 두지 않았다. Stable baselines 라이브러리에서 제공되는 하이퍼 파라미터 값이 기본적으로 사용되었으며, 성공적인 결과를 냈던 기존의 연구에서 사용되었던 값을 그대로 가져온 경우 또한 존재한다. 2.2.1. 절에선 각 알고리즘에 사용한 하이퍼 파라미터의 값이 얼마인지를 표기하였다.

2. 실험 과정

2.1. 실험 설정

실험에선 앞서 언급한 3 가지의 환경에 심층강화학습 알고리즘을 적용해보고, 학습이 진행되었는지를 확인하였다. 다음은 각 환경에 대한 실험 설명이다.

MountainCar 환경의 경우 위에서 설명했듯 discrete 한 환경과 continuous 한 환경의 보상함수가 다르며, 연속적인 환경의 경우 보상설정에 오류가 있어 충분히 빠르게 보상에 도달해보는 exploration 을 수행하지 못하면 높은 확률로 보상을 0 으로 하는(즉, 움직이지 않는) 경우가 최선의 행동이라고 판단하게 된다는 문제가 존재한다. 그렇기에 각 알고리즘을 적용하고 학습이 되었는지에 대한 여부만을 2.2.2. 절에서 설명할 Evaluation Function 과 Learning Curve 을 통해 확인했다.

Lunarlander 환경은 discrete, continuous 환경 둘 다 보상공식이 같기 때문에 세 알고리즘의 학습을 비교분석할 수 있다. Evaluation Function 의 결과를 가지고 각 알고리즘의 성능을 비교하고 학습 곡선을 확인하였다.

CarRacing 환경은 Action Space 가 비대칭이기 때문에 tanh 함수으로 값이 출력되는 DDPG 의 경우 사용이 불가능하였고 DQN 과 PPO 알고리즘만 적용하였다. 모델의 학습에 가장 우려가 되었던 부분은 Delayed Reward 문제이었다. 현재 내가 받은 긍정적인 보상이 언제적 행동으로부터 온 것인지를 알기 어려워 어떤 행동을 강화해야할지 모른다는 의미이다. 환경이 복잡해질수록 이 문제는 더욱 심각해진다. 이를 해결하기 위해 다음과 같은 두 가지 접근을 취했다.

- **Network Structure**: CarRacing 환경은 입력은 96*96 픽셀의 게임의 화면이다. 이를 전부 입력벡터로 받을 시 인공신경망의 노드와 가중치 수가 너무 커지며, 행동에 따른 상태공간의 변화가 상대적으로 매우 작기 때문에 모델이 변화에 집중할 수 없어 학습을 잘 할 수 없다.. 때문에 이를 해결하기 위한 방법은 전처리 또는 Convolutional Neural Network(CNN) 인데,

전처리는 진행하지 않고 대신 신경망 모델로 CNN 을 사용하여 입력 이미지를 모델이 압축적으로 해석하고 변화하는 부분에 집중할 수 있게끔 하였다.

-Frameskip, Framestack: 매 frame 마다 이미지를 입력할 경우, replay buffer 에 있는 이미지가 매우 비슷해지기 때문에 충분한 탐색을 할 수 없게 된다는 문제가 있다. 이를 해 해결하기 위해 입력을 4 frame 마다 받도록 Frameskip 을 사용한 모델과 그렇지 않은 모델을 비교하였다. 또 상태공간의 속도를 표현하기위한 수단으로서 한 번에 4 frame 을 입력으로 받는 Framestack 을 사용한 모델과 그렇지 않은 모델을 비교하였다.

-Action Space Control: 기존 CarRacing 환경이 취할 수 있는 행동 공간은 다음과 같았다.

Steering : [-1(left), + 1(right)]

Gas : [0(no acceleration), 1(fully accelerated)]

Break : [0(without break), 1(fully broken)]

실험 도중 Agent 가 break 행동에 대해서 너무 예민하게 반응하기 때문에 조금만이라도 break 를 강화하는 방향으로 진행이 되면 학습이 이루어지기 힘들어지는 점을 발견하였다. 그래서 행동 공간자체에 제약을 주기위해 Wrapper 메소드를 써서 환경에서 가능한 행동 범위를 제한하였다.

불연속적인 행동 공간 : Break 값을 0 과 0.2 만 줄 수 있도록 설정

연속적인 행동 공간 : Break 값의 범위를 0~0.2 사이로 설정

2.2. 실험의 세부설정 및 결과 확인 방법

2.2.1. 각 알고리즘 별 Hyperparameter

다음은 각 알고리즘 별 최종으로 설정된 Hyperparameter 의 값을 표기한 것이다.

A. DQN

Hyperparameter	Value	Description
Gamma	0.95	the discount factor for future rewards
Buffer size	50000	corresponds to how many experiences should be collected before we do any learning or updating of the model
Batch Size	64	the number of experiences used for one iteration of a gradient descent update
Learning rate	5.00E-04	the strength of each gradient descent update step
Epsilon_Start & End	1, 0.2	start & end of exploration rate
Exploration Fraction	0.1	the exploration level of agent

B. DDPG

Hyperparameter	Value	Description
Gamma	0.99	the discount factor for future rewards
Buffer size	50000	corresponds to how many experiences should be collected before we do any learning or updating of the model
Batch size	128	the number of experiences used for one iteration of a gradient descent update
Actor)Learning Rate	1E-04	the amount of each gradient descent update step
Critic)Learning Rate	1E-03	the amout of each gradient descent update step
Noise	varied	OrnsteinUhlenbeck(OU), AdaptiveParamNoise(Adaparam)
Tau	0.001	the amount of update for target network

C. PPO

Hyperparameter	Value	Description
Gamma	0.99	the discount factor for future rewards
Lambda	0.95	the lambda parameter used when calculating the Generalized Advantage Estimate (GAE)
Buffer size	50000	corresponds to how many experiences should be collected before we do any learning or updating of the model
Batch size	128	the number of experiences used for one iteration of a gradient descent update
Learning rate(optimal)	3.00E-04	the amount of each gradient descent update step

Entropy Coefficient	0.01	entropy coeffcient for the loss calculation
Number of Epochs	10	the number of passes through the experience buffer during gradient descent
Clip Parameter	0.2	corresponds to how much the update will be regulated

2.2.2. 학습의 확인

학습을 확인하는 방법은 다음과 같은 세 가지 방법으로 진행되었다.

A. Evaluation Function

처음 모델을 선언했을 당시의 초기화된 파라미터로 이루어진 모델과 학습된 모델의 성능을 비교해서 학습을 확인한다는 아이디어이다. 평가 함수는 다음 과 같은 형식으로 구성되었다.

```
def evaluate function(model,  num_ steps = 1000):
    :param model: (BaseRLModel object) the RL Agent
    :param num_steps: (int) number of timesteps to evaluate it
   :return: (str) Mean reward for the last 100 episodes  - if the whole episodes are less
  than 100, just return all episodes and show how many episodes it takes.
```

(example return) Mean reward: -196.1 Num episodes: 51

평가함수를 학습 전, 학습 후 실행함으로써, ① 정해진 timestep 동안에 일어난 episode 의 횟수 ② 일어난 각 episode 의 누적된 reward 들의 평균값 을 알 수 있으며, 이 변화를 통해 학습의 여부를 추측할 수 있다.

B. Learning Curves

일반적으로 딥러닝 분야에선 학습의 성공을 오류함수의 수렴으로 판단하나, 심층강화학습에선 학습의 목표가 오류함수의 최소화가 아니라 보상함수의 최대화 이기 때문에 오류함수 만으로는 모델이 잘 학습하고 있는지 확인 할 수 없다. 그 대안으로 학습곡선은 대표적으로 학습의 진행을 모니터링 할 수 있는 좋은 방식이다. 학습 곡선의 x 축은 timestep, y 축은 누적 reward 로 진행한다. MountainCar 와 LunarLander 환경의 경우 pandas dataframe 의 형태로 저장하여 각 episode 별 총 누적보상값과 모든

episode 를 마치는 데 걸린 timestep 에 대한 그래프를 함수를 정의하여 생성하였으며, CarRacing 환경의 경우 각 모델들의 기록을 tensorboard 와 저장하여 학습결과를 시각화하였다.

C. 모델 실행 및 영상 녹화

학습이 된 모델들은 .pkl 형식으로 저장된 뒤 추후 실행을 통해 시각적으로 목표에 달성하는지, 그리고 목표에 달성하기에 적절한 행동을 하는지 등을 확인하였다. 또, 학습하면서 규칙적으로 에피소드를 mp4 파일로 저장하여 학습과정이 어떻게 되는지를 살펴보았다.

[3] 실험결과 및 분석

3. Evaluation Result(timestep = 10000)

3.1 MountainCar

Model	Learning	Mean	Number of Episodes
DQN	Before	-196.1	51
	After	-65.1	74
DDPG	Before	-0.0	11
	After	76.7	43
PPO	Before	-47.2	11
	After	-6.7	11

Evaluation Table 을 보면 DQN 의 경우 학습하기 이전엔 거의 대부분의 경우에 올라가지 못했던 것과 달리(max negative reward 인 -200 에 가까움), 학습 후엔 평균적으로 약 65 번의 timestep 만에 올라가는 데 성공함을 볼 수 있다. 연속형의 경우, DDPG 에 action noise 를 준 모델이 좋은 학습결과를 보여주었다.

3.2. Lunarlander

Model	Learning	Mean	Number of Episodes
DQN (23min.)	Before	-242.7	143
	After	82.8	12
DDPG (50min.)	Before	-120.8	144
	After	-62.8	11
PPO (27min.)	Before	-265.7	89
	After	101.3	13

모든 알고리즘에서 학습 후 한 번의 episode 가 더 많은 timesteps 를 사용하고, mean reward 가 상당히 개선되었음을 알 수 있다. 또, LunarLander 환경의 문제해결 판단 threshold 는 200 점 이상인데, 직접 실행을 통해 확인한 결과 대부분의 경우 200 점 이상을 받는 것을 확인할 수 있었다. 또, 학습이 진행 될수록 각 episode 마다 (0, 0)에 0 speed 로 도착하기 위해 좀 더 세밀한 조정을 하고 더 많은 timestep 을 사용하는 것을 확인 할 수 있었다. 그러나 DDPG 의 경우 눈에 띄는 Mean Reward 의 변화를 찾기 어렵다. 학습 속도 또한 2 배 이상 차이가 나는 것을 볼 수 있다

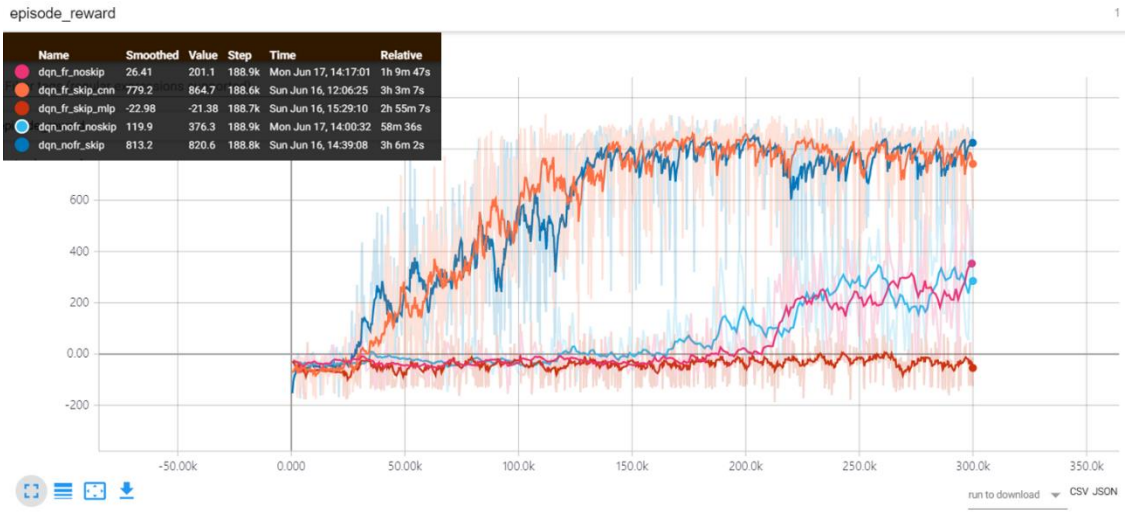
3.3 CarRacing

Model	Architecture	Network	Mean	Number of Episodes
DQN	Framestack, Frameskip	MLP	-19.4	41
	Framestack, Frameskip	CNN	838.2	42
	Framestack	CNN	215.8	11
	Frameskip	CNN	747.5	41
	None	CNN	197.4	11
PPO	Framestack, Frameskip	CNN	-90.7	41
	Framestack	CNN	-71.2	11
	Frameskip	CNN	-90.8	41
	None	CNN	-40.3	11

CarRacing 환경의 경우, DQN 과 PPO 의 결과값의 차이가 확연히 드러났다. CarRacing 환경의 문제해결 threshold 는 600 이며, 여기선 단 두 개의 모델만이 학습에 성공한다. 신경망이 CNN 이고 Frameskip, Framestack 모두 사용한 모델, 그리고 Frameskip 만 사용한 모델이다. 이를 통해 Frameskip 이 학습에 결정적인 역할을 한다는 것을 알 수 있다. Framestack 의 경우 반대로 학습의 영향을 미치지 않는 것으로 드러난다. 또, 같은 Frameskip 과 Framestack 을 사용한 두 모델을 비교해보면, 신경망이 CNN 이냐 MLP 이냐에 따라 학습의 여부가 크게 달라짐을 확인할 수 있다.

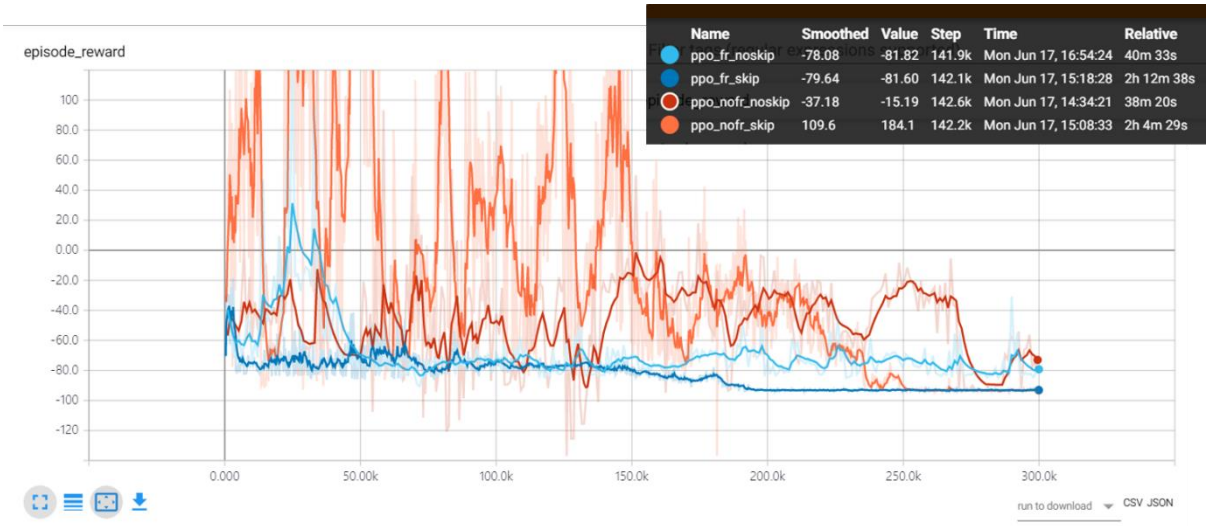
4. Tensorboard 를 이용한 결과 분석

CarRacing 환경은 학습의 여부를 판단하기위해 Stable baselines 모델의 학습을 텐서보드에 저장하여 시각화 하였다. 먼저 DQN 기반 모델 들에 대한 학습곡선 그래프이다.



신경망이 CNN 이고 Frameskip, Framestack 모두 사용한 모델(주황색), 그리고 Frameskip 만 사용한 모델(파란색)의 경우 학습 곡선이 뚜렷한 단조증가의 형태를 보인다. Frameskip 을 적용하지 않거나 신경망이 MLP 인 경우 학습이 매우 느리게 이루어지거나 아예 이루어지지 않음을 볼 수 있다.

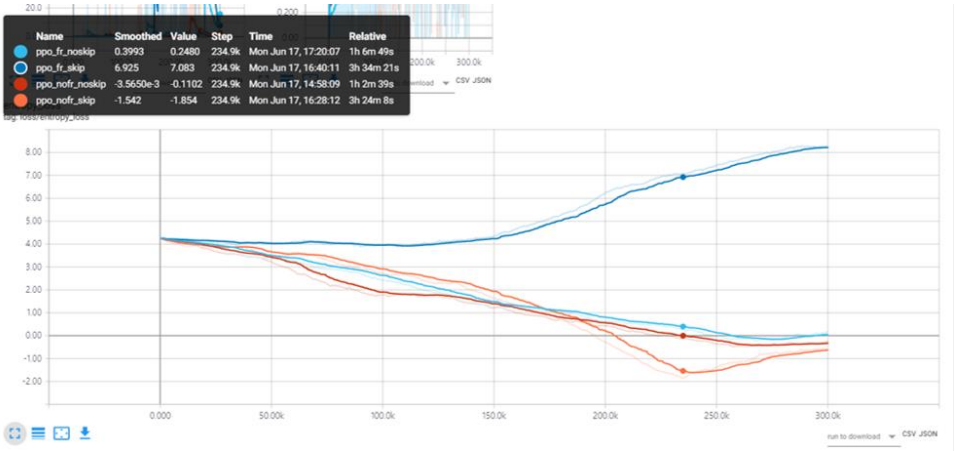
다음은 PPO 기반 모델들의 학습곡선 그래프이다.



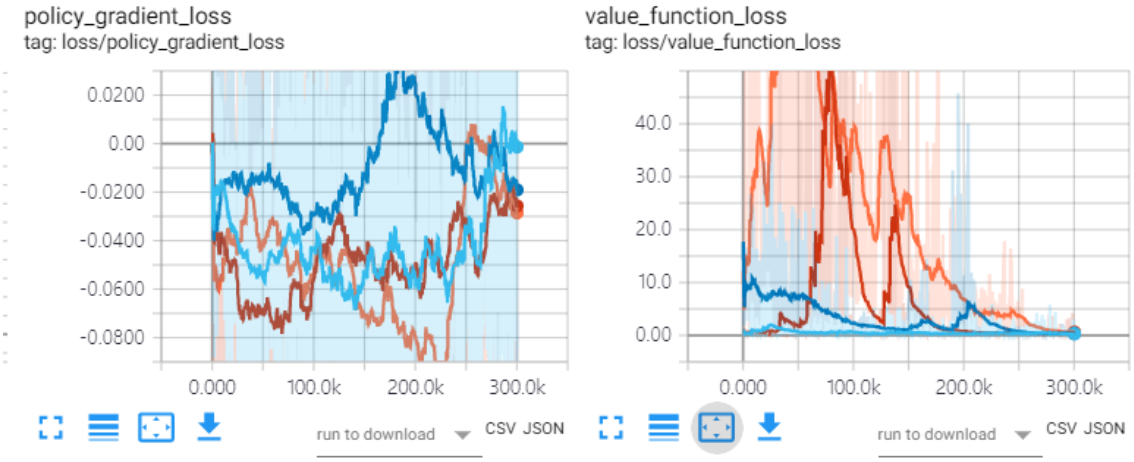
위 PPO 알고리즘의 결과는, 모든 모델들이 제대로된 학습곡선(단조증가)를 그리지 않았으므로 의미있는 비교가 불가능하다. 다만 학습의 결과가 모두 비슷한 정도에서 수렴한다는 것은 알고리즘 모델은 정상적으로 작동되었으나, 어떤 이유에서 잘못된 방향으로 학습이 진행되었음을 시사한다,

이에 보다 심도있는 분석을 위해 학습의 수렴을 파악할 수 있는 다른 요인들을 분석해보았다.

-**Entropy** : 엔트로피는 하나의 행동 확률이 학습에서 너무 큰 비중을 차지하지 않도록, 즉 그로 인해 exploration 이 부족해지지 않도록 방지하는 값이다. 간단히 말해 모델이 얼마나 자주 임의의 행동을 채택하는 지 이다. Entropy 는 성공적인 학습에서 점점 줄어든다. 줄지 않는 다는 것은 학습이 이루어지지 않는 다는 것이고, 0 이 되는 것 또한 같은 상황에서 같은 행동만을 한다는 의미 이므로 좋지 않다. 아래 그래프를 보면 대부분의 경우 점점 줄어드는 경향을 보이나, Framestack, Frameskip 을 사용한 모델의 경우 오히려 증가하는 경향을 보여 학습이 이루어지지 않음을 알 수 있다.



- Policy Gradient/ Value Function Loss : 두 오류함수 값은 정책과 가치함수가 최적의 정책과 가치함수 보다 얼마나 차이가 있는지를 보여준다. Policy Loss 는 음의 값을 취할 수 있지만, Value Loss 는 음의 값을 취할 수 없다. Policy Loss 는 성공적인 학습에서 줄어드는 경향을 보여야하지만, Value Loss 는 받는 보상이 증가할 땐 증가해야하며 이후 optimal policy 에 가까워지면 안정되어야한다. 그래프를 보면 Policy 는 줄어드는 경향을 보이며 최적의 정책을 찾아가는 듯하나, 보상을 전혀 받고있지 않기 때문에 Value Loss 가 0 에 수렴함을 볼 수 있다.



위 두 요인 분석 결과를 보았을 때, PPO 기반 모델들은 모두 학습이 수렴이 되어가고 있으나, Local Maximum 에 수렴하고 있음을 추측할 수 있다. 실제로 학습된 모델들을 실행시켜보면, 모두 움직이지 않고 제자리에 있는 전략을 취함을 확인할 수 있었다. 이는 보상을 더 크게 하는 전략보다 보상의 최소화를 막는 전략이 가장 옳다고 판단한 것으로 보인다.

[4] 결과 토의 및 향후 과제

이번 연구에선 강화학습의 가장 최신 연구 트렌드인 심층강화학습를 알아보고, 심층강화학습이 자율주행차량 에 어떻게 적용되는 지를 조사하였다. 또 가상 자율주행차량 환경 실험을 통해 다양한 환경에 심층강화학습 알고리즘 들을 적용해보고 그 가능성을 알아보았다. 좀 더 복잡하고 연속적인 행동을 하는 자율주행차량 환경에 적용하기 위해서 OpenAI gym 에서 제공하는 환경들에 알고리즘들을 적용하며 그 가능성을 알아보았다.

MountainCar 환경에선 각 알고리즘들의 누적보상값 그래프를 통해 각 알고리즘의 특징과 학습의 여부를 확인하였고, 보상함수의 설정에 따라 학습이 매우 달라질 수 있음을 알 수 있었다. LunarLander 환경에선 알고리즘들의 비교를 통해 어떤 알고리즘이 더 좋은 성능을 보이는지 알아보고자 하였다. 서로 완전히 같은 조건에서 수행된 것이 아니기 때문에 알고리즘의 우열을 가릴 수 있는 척도는 없지만, Mean Reward 의 변화와 Total Learning Time 을 통해 DQN 과 PPO 알고리즘이 상대적으로 우위에 있다는 것을 알 수 있었다.

CarRacing 환경에선 단순히 알고리즘을 적용하는 것만으론 좋은 성과가 나지 않았기에 입력이 이미지로 들어오다는 점을 고려하여 다양한 아키텍쳐를 만들고 비교하였다. 그 결과 DQN 기반의 학습의 경우 입력의 차원의 높은 이미지의 경우 CNN 이 MLP 보다 좋은 성능을 보임을 알 수 있었으며, Framestack 과 Frameskip 방식을 모두 사용한 모델이 가장 높은 누적보상값을 얻음을 확인할 수 있었다. 이 결과는 향후 더 고차원의 행동공간과 상태공간을 가진 실제 환경에서 어떤 방식의 모델이 유용할 지를 알려준다는 면에서 의미가 있다.

그러나, PPO 기반 모델들이 학습이 되지 않은 이유는 해석의 여지가 다양하다. 가장 유력한 해석은 DQN 기반 모델과의 유일한 차이점인 상태공간의 복잡성으로 인해서이다. 일반적으로 Policy Gradient 기반 알고리즘들은 Q 함수에 대한 개선이 아닌 policy 에 대한 점진적인 개선을 채택했기 때문에 optimal policy 가 local maximum 에서 수렴하고 끝낼 경우가 대단히 많다. 여기서도 수렴한 모델들의 경우 학습이 -80 정도의 보상을 받는 local maximum 에서 멈춤을 볼 수 있었다. 추후 local maximum 에서 탈출할 수 있는 방식의 연구가 필요하다

또, 파라메터 튜닝을 진행하긴 하였으나 각 파라메터가 어떤 영향을 미치는 지에 대한 제사한 분석이 진행되지 않았다. 추후 연구에선 다양한 시스템 제어 환경에선 지금보다 더 복잡한 행동공간과 상태공간을 가지고 있을 수 있으므로, 복잡한 상태공간에서 각 변수의 영향과 중요도를 좀 더 알아보아야 할 것이다.

[5] 참고문헌

Advantage Actor-Critic Methods for CarRacing, https://esc.fnwi.uva.nl/thesis/centraal/files/f285129090.pdf

2D Racing game using reinforcement learning and supervised learning, https://github.com/henryteigar/neural-networks-2d-racing-game/tree/car_racing_v0

이재훈, 김태림, 송종규, 임현재. (2018). 가상 환경에서의 강화학습을 이용한 비행궤적 시뮬레이션. 한국시뮬레이션학회논문지, 27(4), 1-8.

이홍석, 박은수, 김승일. (2017). 자율주행자동차 주행을 위한 심화강화학습. 한국정보과학회 학술발표논문집, , 784-786.

채현민, 강창묵, 정정주, 최준원. (2017). 도심주행을 위한 딥강화학습 기반 자율제동 기법. 한국자동차공학회 춘계학술대회, , 625-629.

Plappert, M., Houthooft, R., Dhariwal, P., et al., 2017, PARAMETER SPACE NOISE FOR EXPLORATION arXiv e-prints , arXiv:1706.01905.

Lillicrap, T.~P., Hunt, J.~J., Pritzel, A., et al.₩ 2015, CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING arXiv e-prints , arXiv:1509.02971.

Xi Zhang, Jianqiang, Wang Feng, Zhang, and Keqiang Li, Combining Deep Reinforcement Learning and Safety Based Control for Autonomous Driving arXiv preprint arXiv:161200147, 201

Ben Lau "Using Keras and Deep Deterministic Policy Gradient to play TORCS", https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html

https://gym.openai.com/envs/

https://stable-baselines.readthedocs.io/en/master/index.html