Term Project Paper

강화학습을 이용한 3체 문제의 주기적 궤도운동 찾기.

Find the Periodic Orbital Movement of Three Body Problems Using Reinforcement Learning

물리천문학부 물리학전공

2019-15668 배재용

Abstract

이 연구는 환경에 대한 행동과 보상으로 작동하는 강화학습을 이용하여 3체문제의 주기적 해를 찾고자 진행하게 되었다. 이번 연구에서는 다양한 보상함수 및 환경을 직접 구현하여, 이를 통해 학습된 데이터로 평면에서의 3차운동을 주기적 궤도로 만들게 하는 모델을 구현하는 것을 최종 목표로 하였다. 많은 시행착오 끝에 나타난 연구의 결과로써, 평면에서 매우 작은 오차 내에서 주기적 궤도 운동을 하는 2차원 운동을 만들게 하는 강화학습 환경과 행동, 보상함수를 만들 수 있었다. 이를 통해 입력되는 특정한 조건에 가까운 주기적 궤도 운동의 초기 조건을 찾을 수 있게 되었다.

I. Introduction - Motivation & Purpose

강화학습은 소프트웨어 agent가 스스로의 정책을 바꾸어 가며 높은 보상을 추구하는 머신 러닝 분야 중 하나이다. 가장 대표적인 업적으로는 최근 알파고(Alpha Go)의 바둑 승리 혹은, 그 이후 알파제로(Alpha Star)의 Blizzard 사의 StarCraft에서 프로게이머와의 승리 등을 꼽을 수 있다. 하지만, 이런 높은 성취에도 불구하고 강화학습이 이루어지기 위해 '환경' 및 '보상'의 정의가 명확해야만 높은 성과를 이룰 수 있는 단점이 있다. 그렇기 때문에 대부분 간단한 환경과 보상을 정의할 수 있는 분야에서 많이 다루어 지게 되었다.

강화학습의 알고리즘 큰 틀 자체는 간단하지만, 내부적으로 정의되는 신경망 혹은 보상에 대해 행동을 업데이트 하는 정책은 단순한 코딩으로 구현하기 힘들다. 하지만, 머신 러닝 분야의 발전에 따라, 이미 논문에서 제안된 많은 알고리즘을 가지는 패키지가 많이 존재하고, 이를 통해 performance가 충분히 높은 알고리즘을 사용할 수 있다. 이번 연구에서는 Open Ai gym에서 만든 환경 패키지인 gym및 Facebook 인공지능 연구 팀에서 주로 개발된 *PyTorch Deeplearning library* 를 기반으로 만들어진 알고리즘 및 모델 학습 패키지인 Stable Base-lines3을 사용하였다.

물리의 대부분 문제는 최적화로 생각할 수 있다. 이번에 다룰3체 문제는3개의 물체가 중력 상호작용 하에서 운동을 하는 문제이다. 수치적으로 밖에 풀리지 않아 다양한 근사를 통해 근사해를 찾았지만, 매 순간순간 마다 힘 자체는 계산이 되는 값이기 때문에, 이를 이용해 적절한 환경을 정의하고 '궤도운동'에 대한 적절한 보상을 제시하면 강화학습을 통해 학습시킬 수 있을 것이라 생각되었고, 직접 환경과 보상함수를 설정하여 Agent를 학습시켜 주기적 궤도 운동을 찾는 것을 목표로 연구를 진행하게 되었다.

II. Theoretical Background

1. 강화학습의 기본 이론 및 분류

강화학습은 어떠한 환경 내에서 함수로 정의된 소프트웨어 agent가 현재의 state를 관찰하여 선택하는 행동 중 최대의 보상을 얻을 수 있는지를 학습하는 방법이다.

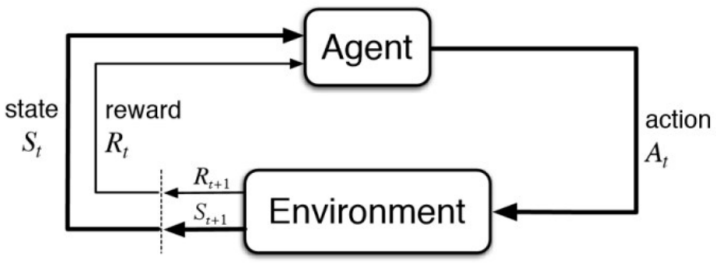

Figure1. Reinforcement learning diagram [Ref:10]

위의 그림과 같이, 소프트웨어 agent(주체)는 상태를 관측하고 선택알고리즘을 기반으로 행동을 결정하게 된다. 강화학습의 agent의 선택 알고리즘은 크게 다음의 세가지의 요소들의 유무로 구분되며 이러한 함수들을 통해 행동을 결정한다.

- Policy Function(정책 함수)

Agent의 행동 패턴으로써, 주어진 상태에 어떤 행동을 할 것인지 결정하는 함수이다. 즉, 상태에 따른 행동을 결정하는

함수이다. 크게 결정적 정책과 확률적 정책으로 나뉘며, 결정적 정책은 주어진 상태에 대해 하나의 행동을, 확률적 정책인 주어진 상태에 대해 행동에 대한 확률 분포를 제공한다.

어떤 agent가 하나의 같은 정책을 통해 학습을 하는지에 따라서 On Policy, Off Policy로 나뉘기도 한다.

$$\text{Deterministic Policy: } a = \pi(s)$$
$$\text{Stochastic Policy: } \pi(a \mid s) = P[A_t = a \mid S_t = s]$$
$$(a: action, s: state)$$

- Value Function(Q-functions) - 가치함수

Agent가 어떤 행동을 결정하였을 때, 그 상태와 행동이 어느 정도의 보상을 얻게 해 줄 것인지를 예상하는 함수이다. 즉, 상태-행동 쌍의 가치를 제공하는 함수이다. 기계학습의 언어로 말하자면, 해당 상태와 행동에 대한 보든 보상의 가중치 합으로 표현된다. 이러한 가중치는 보통 나중의 보상 혹은 직후에 보상에 대한 우선순위를 부여하기 위해 λ(할인계수)를 통해 다음과 같이 표현된다.

$$v_\pi(s) = E_\pi[R_{t+1} + \lambda R_{t+2} + \lambda^2 R_{t+3} + \cdots \mid S_t = s]$$
$$(R: \text{reward}, s: state)$$

- Model

환경에 다음 Step의 상태와 보상이 어떻게 나타날지에 대한 Agent의 예상치를 설정하는 함수이다. State Model과Reward Model로 나뉠 수 있으며 다음과 같이 표현된다.

$$\mathcal{P}^a_{ss'} = P[S_{t+1} = s' \mid S_t = s, A_t = a]$$
$$\mathcal{R}^a_s = P[R_{t+1} \mid S_t = s, A_t = a]$$

이러한 요소들을 통해 알고리즘을 분류할 수 있다.

첫번째로 환경에 대한 Model의 존재 여부에 따라 강화학습을 구분할 수 있다. Model을 정의할 수 있다는 것은 결국 agent가 자신의 행동에 따라 다음환경의 예상이 가능하다는 것이다. 즉, 이러한 Model 하에서 미리 변화를 예상하고 최적의 행동을 시행할 수 있다.

하지만, 보통 강화학습의 환경에서 Model을 보통 정의하기 어렵거나 불가능하기 때문에 Model이 잘 정의된 환경에서 사용할 수 있는 알고리즘과 이외를 구분하기 위해 model-Free와model-based 로 알고리즘을 분류하게 된다.

가치함수(Value function) 및 정책함수(Policy function)에 대해서도 분류가 이루어 진다. 만약 가치함수가 완벽하다면 최적의 정책은 각 상태에서 가장 높은 가치를 가지는 행동일 것이고, 반대로 정책 함수가 완벽하다면 가치함수는 필요하지 않게 된다. 최적의 정책에 따른 행동을 취하면 되기 때문이다.

이처럼 가치함수만을 학습하고 정책은 암묵적으로 가지는 알고리즘을 value-based agent라고 하며, 반대로 정책함수만을 학습하는 알고리즘을 policy-based agent라고 부른다. 각 함수를 학습하는 것에 대한 장단점이 있기 때문에, 가치함수와 정책함수를 모두 가지는 agent도 존재하며 이를 Actor-Critic agent라고 부른다.

이를 통해 다음과 같이 알려진 강화학습 알고리즘에 대한 분류를 하면 다음과 같다.

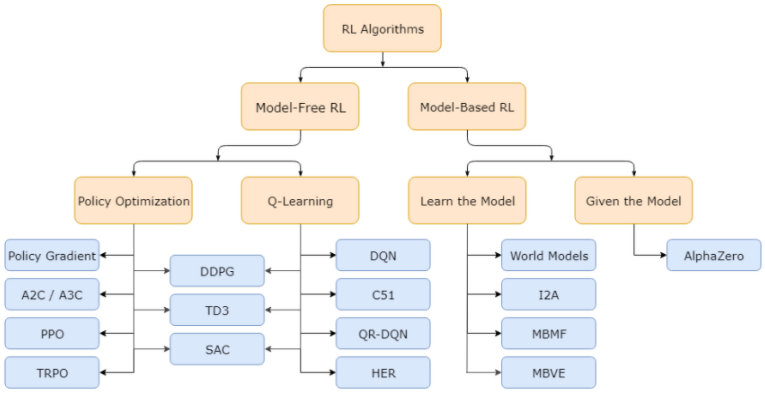

Figure2. A non-exhaustive, but useful taxonomy of algorithms in modern RL [Ref:7]

이번 연구에서는 A2C, SAC, TD3, PPO 알고리즘을 사용하였다.

2. Open AI gym - 환경(Environment) 설정

강화학습을 시행할 환경을 정의하기 위해서, Open Ai gym wrapper을 사용하였다. 강화학습 Agent를 학습시키기 위해 이용한 Stable-baselines3는 Open Ai에서 정의된 gym환경을 사용할 수 있고, gym환경은 다음과 같은 구조로 정의된다.

- $\_\_init\_\_$()

초기화 함수로써, 환경이 최초로 가지는 상태에 대해 정의하는 부분이다. 여기서 강화학습을 위해서는 $Observation\_space$ (관측공간)와 $action\_space$ (행동공간)가 정의되어야 한다. 관측공간은 어떤 순간에서 환경이 가지는 상태의 부분공간으로 정의되며, 행동공간은 Agent가 취할 수 있는 행동의 공간을 의미한다.

- $step$()

입력 받은 행동을 실행하고 4가지의 값을 반환한다.

$Observation$: 새로운 관측 값을 의미한다. 이번 연구에서의 예로 들면3개 물체의 좌표나 주기, 초기속도 같은 값이다.

$Reward$: 이전 행동을 실행함에 따라 결정되는 보상을 의미한다. 보통 이 보상은 내부적으로 다른 함수를 통해 정의된다. 보상의 크기는 정의에 따라 달라지지만, $Agent$의 목표는 항상 총 보상의 증가를 목표로 한다.

$Done$: 환경을 재설정할지 여부를 의미한다. $Done$ 이 $True$ 라면 하나의 에피소드가 종료되었음을 의미하는데, 이는 예를 들어 게임에서 마지막 목숨을 잃는 경우에 해당된다.

$Info$: 디버깅시 유용하게 확인할 정보들을 의미한다. 실질적으로 강화학습이 작동하는 데에는 영향을 끼치지 않지만, 환경이나 보상함수를 수정하기 위한 정보를 얻기 위해 사용된다.

- $reset$()

Done 여부에 따라 에피소드를 초기화 시켜주는 함수이다. Step에 따라 환경의 상태가 바뀌게 되는데, 이를 초기값으로 설정하는 역할을 한다.

-$render$()

환경의 상황을 그래픽으로 출력하는 역할을 한다. 이번 연구에서는 $matplotplib\ \ package$ 을 이용한 단순 궤도그림을 $render()$로 사용하였다.

3. 3체 문제의 운동방정식

3체 문제도 결국 힘의 관점에서 보면 단순한 만유인력이 작용하는 물체의 운동일 뿐이다. 이는 다음과 같이 표현할 수 있다.

$$m_i \frac{d\vec{v}_i}{dt} = \frac{G m_i m_j}{r_{ij}^3} \vec{r}_{ij}$$

$$\frac{d\vec{r}_i}{dt} = \vec{v}_i$$

이를 수치적으로 적분하여 각 물체의 위치를 구할 수 있다.

III. Research Contents - Research procedure & Results

모든 과정은 python으로 이루어 졌으며, package에서 제공한 함수 이외의 모든 코드는 직접 작성하였다.

기본적인 환경 설정에서 행동 공간 및 Step 함수의 구현은 다음과 같이 이루어 졌다. 만유인력 상수 G는 1로 계산에 사용하였고, 모든 물체의 질량은 1로 동일하게 주었다.

Action Space(행동공간) -	3체의 초기 위치 혹은 속도 등 과 같은 초기 조건에 가할 변화량
Gym - Environment 구조의 Step 함수	State - 3체의 초기조건을 이용한 수치적분 해 - 3체의 시간에 따른 위치 Reward - 3체의 시간에 따른 위치 정보를 이용한 보상함수 정의

Done - 3체의 시간에 따른 위치 정보에 따른 에피소드 종료 여부 정의
Observation - 3체의 전체 위치 정보의 부분 공간을 관측 공간으로 정의

연구에 사용된 기본적인 행동 공간 및 Open AI gym 환경 구조

예를 들면 다음의 과정이 일어난다. $\_init\_()$ 에서 최초로 가지는 3체의 위치를 각각 $\overrightarrow{r_{1,0}}$, $\overrightarrow{r_{2,0}}$, $\overrightarrow{r_{3,0}}$ 이라고 하자, 또한 행동 공간이 $[Actionspace: [\Delta x1, \Delta y1, \Delta z1],\ each \pm 1/100]$ 와 같이 정해진 경우, 각 Step마다 가지는 3체 초기 위치는 $\overrightarrow{r_{1,0}} + (\Delta x1, \Delta y1, \Delta z1)$, $\overrightarrow{r_{2,0}}$, $\overrightarrow{r_{3,0}}$ 로 주어지게 된다. 이 초기조건을 통해 수치적분을 진행하고, 보상함수를 통해 보상을 받는다.

1. 3차원 3체문제 RL 구현 시도.

가장처음 환경을 정의할 때 사용한 방법은, 3체 문제에서 한 물체의 초기 위치를 움직여 가며 궤도 운동을 찾는 방식을 사용하였다. 이번 시도에서 사용한 기본적인 설정은 다음과 같다.

$$initial\ position: 1: (0,0,0), 2: (-1,0,0), 3: (1,0,0), velocity\ is\ all\ zero$$

$$Actionspace: [\Delta x1, \Delta y1, \Delta z1], \quad each \pm 1/100$$

$$Done: Collision$$

$$Observation: Initial\ position$$

이를 기본적으로 설정한 다음 보상함수에 대한 고려를 통해 강화학습이 적절하게 이루어 지도록 시도하였다.

A1. Fourier변환 및 함수 fitting을 통해 '궤도'적 운동 여부 결정 보상함수 정의

처음 시도한 방법은 Fourier변환을 이용한 방법이었다. 목표로 하는 주기적 '궤도'의 여부를 확인하기 위해서, 초기조건을 이용해 시간에 따른 3체의 위치를 계산한 후 각각 좌표가 유사-궤도 운동을 하는 만큼 보상을 주는 방안을 생각해 보았다. 3체의 위치를 수치적분으로 계산할 때, 사용한 시간 간격과 최종 시간을 각각 ΔT, T_f 라고 하자. ($\Delta T = T_f / n$)

즉 각 3체의 좌표 $r_i(t) = (x_i(t), y_i(t), z_i(t))$ 를 각각 FFT를 진행한 다음, 여기서 높은 피크를 기준으로 예상 주기를 정하고, 이를 통해 각 좌표를 Sin함수에 fitting 했다. 예상 주기($T_{expected}$)를 이용해 Fitting된 함수를 이용하여 다음과 같은 보상 정책을 설정하였다.

1. 주기 보상: $\left| T_f \,/\, T_{expected} \right|_{i,j}\ i = (x, y, z), j = (1,2,3)$ 의 평균.
2. R^2보상: Fitting 된 함수와 실제 좌표사이의 결정 계수 값 $R^2_{i,j}\ i = (x, y, z), j = (1,2,3)$ 의 평균

(각 보상은 동일한 order을 가질 수 있도록 가중치가 곱해져 최종 보상에 적용되었다.)

이를 통해 단순 발산 운동이기 때문에 Fitting에서 너무 큰 예상 주기를 가지는 경우 적은 보상을 주게 하였고, Fitting이 잘 될수록 높은 보상을 Agent에게 가하였다.

이를 통해 적절한 예상 주기를 가지고 높은 결정계수로 Sin함수에 잘 Fitting되는 궤도 운동이 agent에 의해 결정될 것이라 예상하였다.

B1. Results

A2C, SAC, TD3, PPO의 알고리즘을 이용해 5000, 10000회 학습을 진행하였고, 학습된 Agent를 환경에 적용한 후, 1000 Step 진행시켰다.

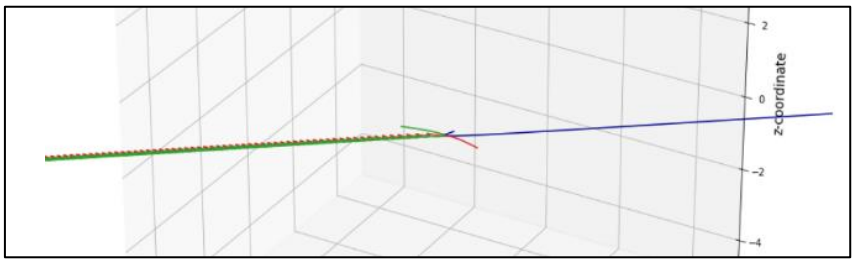

Figure3. 대표적인 실패 사례. 주기보상과 결정계수 보상.

결과는 대부분 다음과 같은 초기조건 - 궤도를 보여주었다. Agent는 적절한 궤도 운동을 위한 행동의 조건을 학습하지

못하였다. 즉, 위와 같은 보상 함수를 통해서는 적절한 궤도 운동이 찾아지지 않았다. 알아낸 문제점은 다음과 같다.

1. R^2 보상은 보상으로써 유의한 값의 차이를 나타내지 않았다. Python $Scipy\ package$의 fitting 함수 자체가 최적 값을 제안하기 때문에, 어떠한 이상한 궤도 상에도 강제로 맞춘 Fitting 함수와의 R^2는 0.9정도로 나타났다. 하지만 더 큰 문제점은 궤도의 주기성과 상관없이 높은 값을 낼 수 있는 보상 정책이라는 점이다.

2. 범위 제한의 유무로 인해, 매우 많은 학습 이후 결과값들이 마치 발산하는 모습을 보여주었다. 매우 큰 시간 범위에서는 궤도 운동의 일부로 고려할 수 있을 수 있으나, 무한한 시간에 대한 연산을 못하기 때문에 3체의 초기 조건 및, 움직일 수 있는 가능 공간의 제한을 주는 것이 요구되었다.

A2. 보상함수 항 추가 및 Done 조건 추가

위의 문제점을 막기 위해 보상함수 및 Done 조건을 추가하였다. 개선된 보상 함수는 다음과 같다. FFT 및 Sin fitting으로 구해진 함수(3체의 위치에 대한 Fitting 함수)를 $r(t)_{i,j}\ \ i=(x,y,z), j=(1,2,3)$, 실제 수치적분으로 구해진 3체의 좌표를 $r_{int}(t)_{i,j}$라고 하자.

1. D^2err 보상: $-\left|r(t)_{i,j}-r_{int}(t)_{i,j}\right|_{i,j}\ i=(x,y,z), j=(1,2,3)$ 의 평균.

2. $range$ 보상: $r_{int}(t)_{i,j} > range$ 인 경우마다 감점

3. R^2보상 삭제

(각 보상은 동일한 order을 가질 수 있도록 가중치가 곱해져 최종 보상에 적용되었다.)

err항은 적을수록 높은 보상이어야 하므로 음의 부호를 택하였다.

B2. Results

A2C, SAC, TD3, PPO의 알고리즘을 이용해 1000, 5000회 학습을 진행하였고, 학습된 Agent를 환경에 적용한 후, 1000 Step 진행시켰다.

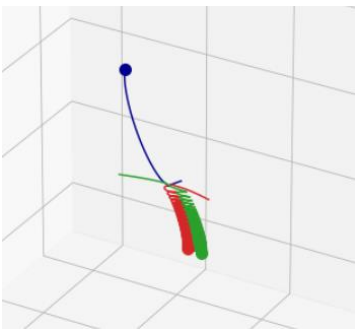

Figure4. 대표적인 실패 사례. 주기보상과 D^2err 보상, $range$보상.

이번에도 agent가 정책을 학습하지 못하였다. 이러한 대표적인 실패 사례 외에도, 또다시 범위를 벗어나거나 하는 문제가 발생했다. 단순한 $range$ 보상으로는 궤도가 발산하는 것을 막을 수 없을 뿐 아니라, 위와 같은 추가적인 보상함수를 통해서도 주기적 궤도 운동을 학습하지 못했다.

2. 2차원 3체문제 RL 구현 시도 - 초기값 변수에 운동량 추가.

시간의 제한과 컴퓨터의 사양상의 문제로, 3차원에서 연구를 진행하는데 어려움을 느끼게 되었다. 이에 차원을 내려 환경 및 행동 공간, Step 함수 설정 수정을 계속 진행하였다.

이전 절에서, 행동 공간이 하나의 물체만 움직이게 정의하였었다. 하지만 이는 조건 하에서 주기적 궤도 운동이 있다고 하더라도, 너무나 제한적인 초기조건을 의미하였다. 그렇기 때문에 이번 시도에서는 다음과 같은 기본적인 설정을 사용했다.

$$initial\ position: 1:(0,0), 2:(-1,0), 3:(1,0) \quad initial\ velocity: 1:(0,0), 2:(0,1), 3:(0,-1)$$

$$Actionspace:\ [\Delta x2, \Delta y2, \Delta x3, \Delta y3, \Delta v_x 2, \Delta v_y 2], \quad each \pm 1/100$$

$$Done: Collision$$

$$Observation\ :\ Initial\ position, velocity$$

즉 행동 공간을 2번 3번 물체의 x, y 초기 좌표 및 2번 물체의 초기 속도로 정하였다. 또한, 3번 물체의 초기 속도는 $-v_2$로 설정하였다.

위의 2-B1에서 고려한 보상함수를 통해 학습을 진행하여도, 원하는 궤도 운동은 나타나지 않았다. 즉 적절한 '주기적 궤도'를 만족시키게 agent의 학습되지 않았다. 머신 러닝의 특성상, 원하지 않는 방향으로 학습이 진행되는 경우 agent의 학습 '경로'를 알 수 없기 때문에 무엇이 문제인지 정확히 특정 지을 수는 없지만, 차원을 내리고 행동 공간을 바꾸어 보아도 비슷하게 실패한 궤도 운동이 나타나는 것은 보상함수에 문제가 있다고 판단되었다.

3. 2차원 3체문제 RL 구현 시도 - 초기값 변수에 운동량 및 주기 추가

A1. 보상함수 및 행동공간의 개편 - 주기 추가 및 보상함수 단순화.

행동 공간에 agent가 특정 주기를 선택할 수 있게 설정함으로써, 보상함수를 단순하지만 강력하게 조정하였다. 새롭게 정의된 행동 공간과 관측공간은 다음과 같다. 초기 위치와 속도는 동일 값을 사용하였다.

$$initalperiod: T_0 = n/2 \qquad where(\Delta T = T_f/n)$$

$$Actionspace: [\Delta x2, \Delta y2, \Delta x3, \Delta y3, \Delta v_x 2, \Delta v_y 2, \Delta n], \qquad position\ \&\ velocity \pm 1/100\ , n = -1, 0, 1$$

$$Done: Collision$$

$$Observation: Initial\ position, velocity\ and\ selected\ period$$

가장 기본적인 주기함수의 조건은 $f(t+T) = f(t)$ 임을 만족하는 것이다. 지금까지는 주기 자체를 미리 특정 지을 수 없었기 때문에 이러한 간단한 정의를 쓰지 못하고 FFT & Sin fitting을 이용하였지만 주기를 행동 공간에서 설정할 수 있는 경우 간단한 해석이 가능해진다.

즉 다음과 같은 보상함수를 적용하였다. 수치적분을 통해 구해진 각 물체의 좌표를 $r_{int}(t)_{i,j}$ $i = (x, y)$, $j = (1,2,3)$ 라고 하자.

1. $Error\ Reward$: $-\sum_t^{T_f} \left| r_{int}(t)_{i,j} - r_{int}(t + T_0 + \Delta n)_{i,j} \right|$, $t \in [0, T_f/n, \dots, T_f]$ 의 i 에 대한 평균.
2. $Range\ reward$: $-|r_{int}(T_f)_{i,j}|$ 의 평균.

(각 보상은 동일한 order을 가질 수 있도록 가중치가 곱해져 최종 보상에 적용되었다.)

두 경우 모두 적으면 좋은 주기적 궤도운동을 예상할 수 있으므로 음의 보상 정책을 사용하였다.

B1. Results

A2C, SAC, TD3, PPO의 알고리즘을 이용해 10000, 12000, 30000회 학습을 진행하였고, 학습된 Agent를 환경에 적용한 후, 1500 Step 진행시켰다.

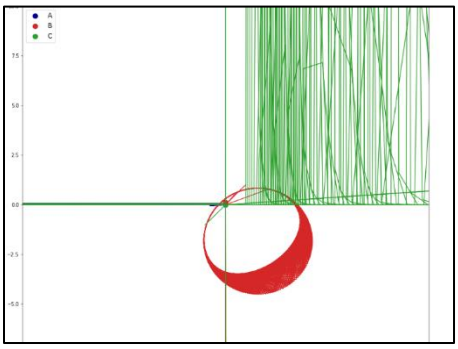

Figure5. 대표적인 실패 사례. err 보상, $range$보상. 각 물체당 점 20000개 도시.

이번 경우 또한 결과 자체는 실패로 돌아갔다. 하지만 괄목할 만한 성과가 이루어 졌다. 나머지는 매우 이상한 결과가 나타났지만, B만은 궤도 운동을 보였다. 이를 Agent의 관점에서 보면, 최소 하나의 개체의 운동은 궤도 운동으로 해야만 좋은 보상이 나온다는 것을 인식했다는 것이다.

하지만 처음부터 보상함수를 통해 제어하고자 했던, $range\ limit$ 이상으로 가는 것을 온전히 막지는 못했다. 이는 역시

보상함수의 완전치 못한 정의에서 야기됐다고 생각한다. 지금까지, 보상함수를 정의하면서 같은 order을 맞추거나, 추가적으로 $range\ limit$ 에 대한 더 큰 가중치를 부여한 경우에도 $range\ limit$ 를 마구잡이로 벗어나는 운동이 많이 관측되었다. 이를 통해 보상함수를 통한 추가적 제어의 문제점을 다음과 같이 생각 할 수 있다.

1. 보상함수를 다양한 경우로 정의할 경우, agent는 행동에 따라 크게 변하지 않는 하나의 보상함수를 거의 무시하고, 나머지 보상함수에 따른 보상을 늘리기 위해 정책을 수정한다.

2. 다양한 보상 정책으로 보상함수를 정의할 경우, 어떤 가중치를 쓰더라도 Normalize와 같은 변형은 거의 불가능 하기 때문에, 1번에 의한 영향을 쉽게 제거할 수 없다.

이러한 문제점 때문에 '다양한 보상함수를 통한 agent 제어' 라는 방법이 맞지 않음을 확인할 수 있었다. 즉, 보상함수는 가능한 한 함수로 정의되어야 agent가 적절한 방향으로 정책을 업데이트할 수 있다는 결론을 얻을 수 있었다.

A2. 보상함수의 단일 정책화 및 Done을 통한 $range\ limit$ 제어.

완전히 $range\ limit$를 시행할 수 있는 방법은 특정 제한점을 초과하는 운동이 발견되는 순간 그 에피소드는 완전히 틀린 정책과정이라고 agent에게 알려주는 것이다. 결국 다음과 같은 done 정책과 행동 공간을 사용하였다.

$$initalperiod: T_0 = n/2 \qquad where(\Delta T = T_f/n)$$

$$Actionspace: [\Delta x2, \Delta y2, \Delta x3, \Delta y3, \Delta v_x2, \Delta v_y2, \Delta n], \quad position\ \&\ velocity \pm 1/100\ , n = -1, 0, 1$$

$$Done: Collision\ , range\ limit, period\ limit$$

$$Observation: Initial\ position, velocity\ and\ selected\ period$$

모든 순간마다 충돌 여부와 $range\ limit$를 확인할 뿐 아니라, 오류를 없애고 비 정상적인 주기 운동(주기가 0 이거나, 매우 클 때($T_0 \gg n$)의 운동)을 제한하기 위해 추가적으로 $period\ limit$를 $done$ 정책에 추가하였다.

보상함수는 정해진 주기에 따라 결정되는 오차항만을 고려하였다.

$$Error\ Reward\ : -\sum_t^{T_f} \left| r_{int}(t)_{i,j} - r_{int}(t + T_0 + \Delta n)_{i,j} \right|,\ t \in [0, T_f/n, \dots, T_f]$$ 의 i 에 대한 평균.

B2. Results

A2C, SAC, TD3, PPO의 알고리즘을 이용해 10000회 학습을 진행하였고, 학습된 Agent를 환경에 적용한 후, 1500 Step 진행시켰다.

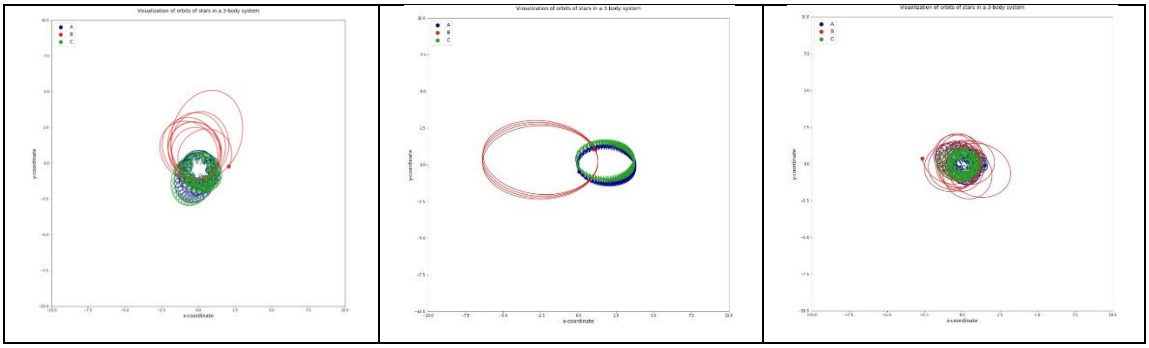

Figure6. 좌측부터 SAC, PPO, A2C의 10000회 Agent 학습 후 1500step 진행 시 가장 높은 보상의 궤도. TD3의 경우 에피소드가 진행되는 Step이 없어 도시하지 않았다.

이전의 결과와 비교했을 때, 눈에 띄게 주기적 궤도운동과 비슷한 운동이 이루어 졌음을 확인할 수 있었다. 특히 PPO algorithm을 이용한 agent의 학습결과가 매우 놀라운 결과를 보여주었다.

다시금 PPO의 알고리즘을 이용해 25000,30000,50000회 학습을 진행하였고, 학습된 Agent를 환경에 적용한 후, 5000 Step 진행시킨 후, 가장 reward가 높은 궤도를 도시하면 다음과 같다.

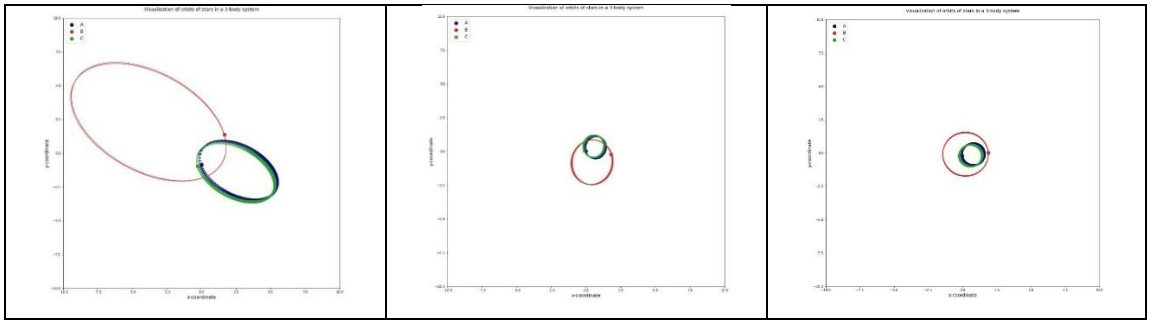

Figure7. 좌측부터 $25000, 30000, 50000$회 agent학습 후 5000Step 시 가장 높은 보상의 궤도.

학습량에 따른 차이는 크지는 않았다. 또한, 같은 학습된 agent라고 해도 가장 처음 시행하는 행동은 무작위적 특성을 가지고 있기에, 상이한 결과가 나오는 것은 당연하다고 볼 수 있다.

각 15000회 Step을 진행하는 동안 1500Step 이하에서 가장 reward가 높은 주기적 궤도가 관측되었다. 이는 아직까지 agent가 $done$ 정책을 완전히 피하면서 학습이 된 것은 아니라고 볼 수 있다. 하지만 이는 당연한 결과라고 할 수 있다. Agent의 학습 단계에서, $done$ 정책이 $True$ 가 되는 순간 그 에피소드는 종료하기 때문에, 그 다음 상태에 대한 '복구' 정책은 학습할 수 없는 것이 당연하다. $done$ 정책을 이번의 경우처럼 단순 상수로 규정짓지 않고, 초기 조건에 대한 적절한 $range\ limit$ 정책[1]으로 재구성한다면 더 나은 Agent를 구현할 수 있을 것이라 생각된다.

IV. Results & Conclusion[2]

1. Results: RL Agent를 통한 2차원 3체 문제의 주기적 궤도 조건 발견

최종적인 행동 공간과 보상 함수 및 $done$ 정책을 통해 PPO algorithm 하에서 높은 정확도의 주기적 궤도 운동을 구현해 낼 수 있었다.

학습 자체는 초기조건과 무관하게 진행되었기 때문에, 환경에서 최초로 정의한 3체의 위치와 초기속도, 초기 주기 값을 수정하여도 비슷한 결과를 얻을 수 있다.

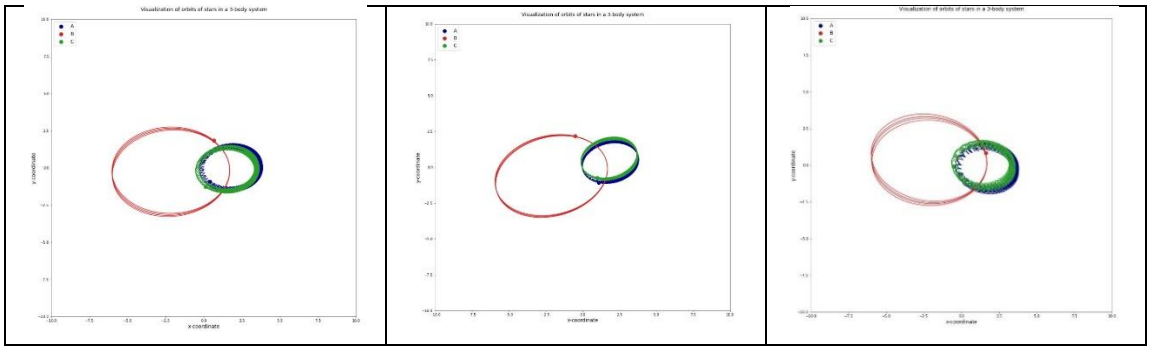

Figure7. 좌측부터 초기조건 $T_0 = n/4$ 로 설정한 경우 구해진 궤도, 초기조건, $T_0 = n/4$, $r_2 = (2,0)$ 으로 설정한 경우 구해진 궤도. $T_0 = n/4$, $r_2 = (2,0)$, $v_3 = (1,1)$ 으로 설정한 경우 구해진 궤도.

2. Conclusions & Furthermore:

---

[1] $range\ limit$를 완전히 제거할 수는 없다. 왜냐하면, 이를 없애는 경우 단순히 3체를 모두 매우 멀리 위치하게 만들어 거의 이동하지 않게 만들면, 위치가 상수함수 꼴이 되고 이는 모든 주기함수를 만족하게 되기 때문이다. 이 경우도 연구 과정 중 나왔지만 제한상 모든 시행착오를 적지는 못하였다.

[2] 연구에서 사용한 모든 코드는 $Colab$을 이용하였다. 코드확인을 위한 주소는 다음과 같다.
https://colab.research.google.com/drive/1h8u5yz50Lv2UCo6T7aOgkmho8ZS5o6u5?usp=sharing

이번 연구는 강화학습의 3체문제 적용이라는 단순한 생각으로부터 시작되어 결론적으로 2차원에서 주기적인 궤도를 찾는 Agent 및 환경, 보상 정책, 행동 정책을 구형해 내는 것을 성공하였다. 결과는 인상적이었지만 아쉬운 점이 몇 가지 있다.

첫번째로, 3차원에서의 3체 문제 주기적 운동 구현을 시도하지 못하였다. 이는 여러 이유가 있지만, agent가 학습을 올바르게 진행되는지 확인하기 위해서는, 많은 Timestep 하에서의 학습이 불가피한데, 보상함수 수정- 학습 (5000~10000회)-결과 확인 (1000회)의 과정 중 agent를 학습시키는 과정이 거의 10시간 정도 걸리는 경우도 있어, 시간의 제약이 크게 다가왔다. 프로젝트 시작은 약 한 달 전부터 조금씩 시작했으나, 환경 및 내부 정책들을 수정하고 확인하는 과정을 최대 하루에 2~3번 밖에 진행할 수 없었고, 이에 3차원 구현에 대한 결과를 얻을 수 없었던 것이 아쉽다.

특히, 2차원 구현을 하는 도중 알게 된 C-언어 $ODEint - wrapper$인 jet package를 미리 알고 진행하였으면 충분한 시간이 났을 것이라는 아쉬움이 남는다.

이번 연구는 Logistic map이 Chaotic함을 알기 위해 사용한 bifurcation diagram과 Cobweb diagram으로부터, Chaotic한 구간이 있기 위한 함수의 개형(위상적 성질)에 대한 이해를 높여가면서 진행되었다. 이를 통해 Chaotic한 양상을 가지기 위한 제한 조건을 찾아냈고, 다양한 함수에 적용하면서 실제로 유용성을 확인하였다.

하지만, 이번 연구에서도 시사하는 바는 적지 않다. 이번 연구를 통해서 보상함수의 적절한 설정 방안이나, 최적화하기 위한 $done$ 정책 수정과 같은 물리계에서 사용할 수 있는 RL 알고리즘 - 환경 설정에 대한 적절한 방향을 알 수 있었다. 또한, 구현된 RL 알고리즘을 통해 매우 적은 오차 내에서 주기적인 운동을 하는 것을 관측할 수 있었던 결과 자체도 매우 의의가 있다고 본다.

또한, 결과에 오차가 있는 경우, 대부분 세차운동의 결과로 나타났다. 그렇다면 세차운동에 대한 보상함수를 다시금 정의한 다음 1차적인 Agent 학습 결과로 나온 초기조건을 학습된 세차운동-Agent에 넣어 결과를 얻는다면, 더욱 정확한 주기적 궤도 운동을 2가지의 RL Agents를 통해 얻을 수 있을 것이라 기대되며, 이는 추후 다시금 연구해 볼 것이다.

이전 논문을 참고하더라도, Grid-Search 방법 및 복잡한 수식을 통해 단순 기계적 최적화의 관점에서 주기적 궤도를 찾거나[ref:9], 혹은 단순 DNN(Deep Neural Network)및 복잡한 수식적 고려항을 통하여 주기적 궤도를 찾았는데[ref:1] 강화학습을 이용한 문제 해결에 대해서는 이루어진 연구가 없었다. 이전의 연구보다 훨씬 단순한 수식으로 정의된 보상함수와 RL algorithm Agent를 통해 궤도를 찾는 본 연구는 정확도는 상대적으로 낮지만, 간단한 수식에 비해 매우 적은 오차를 가지며, 확장되기 쉬운 장점을 지닌 새로운 컴퓨팅 방법을 제시했다는 큰 의의를 지녔다고 볼 수 있다.

V. References

1. Breen, P. G., Foley, C. N., Boekholt, T., & Zwart, S. P. (2019). Newton vs the machine: solving the chaotic three-body problem using deep neural networks. Retrieved from http://arxiv.org/abs/1910.07291
2. Deshmukh, G. (2019, July 2). Modelling the Three Body Problem in classical mechanics using python. Retrieved November 28, 202 0, from Towards Data Science website: https://towardsdatascience.com/modelling-the-three-body-problem-in-classical-mechanics-using-python-9dc270ad7767
3. Fujimoto, S., van Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. Retrieved from http://arxiv.org/abs/1802.09477
4. Géron, A., & O'Reilly Media. (2020). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems (6th ed.). Heidelberg, Germany: O'Reilly.
5. Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). So ft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. Retrieved from http://arxiv.org/abs/1801.01290
6. Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., … Kavukcuoglu, K. (2016). Asynchron ous methods for deep reinforcement learning. Retrieved from http://arxiv.org/abs/1602.01783
7. Part 2: Kinds of RL Algorithms — Spinning Up documentation. (n.d.). Retrieved November 28, 2020, from Openai.com website: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html
8. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. Retrieved from http://arxiv.org/abs/1707.06347
9. Šuvakov, M., & Dmitrašinović, V. (2014). A guide to hunting periodic three-body orbits. American Journal of Physics, 82(6), 609–619.
10. Bhatt, S. (n.d.). 5 things you need to know about Reinforcement Learning - KDnuggets. Retrieved November 28, 2020, from Kdnuggets.com website: https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html