ISSN 3022-8719

Journal of Science and Science Education | Vol. 49, No. 2, p. 66-74, December 2024

https://doi.org/10.23335/JSSE.2024.49.2.66

# 인공 지능 강화 학습 개념과 실험 데이터 분석에 응용

**정은석 · 한상욱***

전북대학교 과학교육연구소, 전북대학교 융합과학연구소, 전북대학교 사범대학 과학교육학부 물리교육전공, 전주 54896

# Concept of Artificial Intelligence Reinforcement Learning and Applications to Experimental Data Analysis

**Eun-Suk Jeong and Sang-Wook Han***

Institute of Science Education, Institute of Fusion Science, and Division of Science Education, Department of Physics Education, Jeonbuk National University, Jeonju 54896, Korea

**초 록**: 인공 지능 강화 학습은 환경과 에이전트 간의 상호작용을 통해 순차적인 문제를 해결하며, 보상을 기반으로 에이전트를 학습시키는 방법이다. 이것은 인공 신경망과 강화 학습을 결합한 심층 강화 학습으로 머신러닝의 지도 학습과 비지도 학습의 한계를 극복할 수 있는 가능성을 제시한다. 본 논문에서는 다이나믹 프로그래밍을 활용한 강화학습의 정책 반복 학습 과정을 기술하였다. 벨만 방정식으로부터 유도된 가치 함수 (Value Function)와 Q-함수가 그리드 월드 (Grid World) 환경에서 어떻게 적용되는지를 기술하여 강화 학습의 기본 개념을 상세히 설명하였다. 또한 심층 강화 학습 방법 중의 하나인 A3C (Asynchronous Advantage Actor-Critic) 알고리즘을 EXAFS (Extended X-ray Absorption Fine Structure) 데이터 분석에 적용하여 심층 강화 학습이 과학 데이터 분석에 어떻게 활용될 수 있는지를 기술하였다.

**중심어**: 강화 학습, 머신러닝, 딥러닝, 인공지능

**Abstract:** Reinforcement learning (RL) is a method that addresses sequential decision-making problems by enabling an agent to interact with an environment and learn from rewards. Deep RL, a fusion of artificial neural networks and RL, shows promise in surpassing the constraints of supervised and unsupervised learning in machine learning. This study delves into the policy iteration learning process of RL using dynamic programming. It elaborates on how the value function and Q-function, derived from the Bellman equation, are leveraged in a Grid World environment to elucidate the core tenets of RL. Furthermore, practical applications of deep RL are showcased through the utilization of the A3C (Asynchronous Advantage Actor-Critic) algorithm in the analysis of Extended X-ray Absorption Fine Structure (EXAFS). This demonstration underscores the effective integration of deep RL in scientific data analysis.

**Keywords:** Reinforcement learning, Machine learning, Deep learning, Artificial intelligence, EXAFS

*Corresponding Author: Sang-Wook Han
Phone: +82-63-270-2775
Email: shan@jbnu.ac.kr

## I. 서 론

머신러닝은 입력 데이터로부터 패턴을 학습하고 이를 이용하여 예측 또는 의사결정을 하는 기술로 21세기 다양한 산업 분야에서 혁신적인 변화를 이끌고 있다. 2024년 노벨물리학상 분야에서 물리분야[1]와 화학분야[2] 모두 머신러닝에 대한 기여로 수상된 것은 머신러닝의 중요성을 보여준다. 의료분야[3,4]에서 머신러닝 기반 진단 시스템은 질병을 조기에 감지하고 개인 맞춤형 치료를 가능하게 하였으며 금융 분야에서는 거래 패턴 분석으로 부정거래를 탐지하는 등 우리 생활에 깊숙이 들어와 활용되고 있다[5,6]. 그림 1(a)와 같이 머신러닝은 크게 지도 학습(Supervised learning), 비지도 학습(Unsupervised learning), 그리고 심층 강화 학습(Deep reinforcement learning, DRL)으로 나누어진다[7,8]. 지도 학습은 입력된 데이터와 정답이 주어진 상태에서 인공 신경망을 학습하고 새로운 데이터를 훈련된 인공 신경망에 넣어 새로운 결과를 예측한다. 반면 비지도 학습은 정답이 없는 상태에서 패턴을 발견하는 학습 방법으로 데이터 군집화와 차원 축소 등에 주로 사용된다. 주로 과학 분야에서 사용되는 학습은 지도 학습으로 생명과학 분야의 알파폴더가 대표적인 예이다. 생명과학 분야에서는 단백질의 구조에 대한 빅데이터를 가지고 있어 훈련된 신경망으로 새로운 모델을 만들어낸다. 이 외에도 다양한 분야에서 지도 학습을 이용하여 데이터를 분석하고 새로운 결과들을 찾기 위해 많은 노력을 기울이고 있다[9,10]. 하지만 지도 학습은 신경망 훈련을 위하여 사전에 많은 양의 데이터를 확보해야 하고 각 데이터의 정답을 제공해야하기 때문에 많은 양의 데이터와 정확한 정답 확보에 엄청난 비용과 시간이 필요하다. 더군다나 실제 과학 분야에서는 신경망 훈련에 사용된 것과 전혀 다른 새로운 환경이나 정답이 준비되지 않은 데이터들이 많기 때문에 지도 학습 방식을 과학 분야에 적용하는 데는 한계가 있다. 이를 극복할 수 있는 방안으로 심층 강화 학습이 더욱 주목받고 있다. 심층 강화 학습은 데이터에 대한 정답이 요구되지 않으며 주어진 데이터를 자동으로 탐색하여 보상 함수로 정답을 찾는다. 새로운 환경 또는 시스템이 주어진다할지라도 이전 시스템을 기반으로 보상을 잘 받는 방향으로 시스템을 변화시킬 수 있다는 점은 새로운 상황 또는 새로운 물질이 발견되는 과학 분야에서 유용하게 사용될 수 있음을 시사한다. 심층 강화 학습은 강화 학습과 인공 신경망을 결합한 기법으로 에이전트가 환경과 상호작용하며 보상을 최대화하도록 학습한다. 그림 1(b)는 실험쥐가 강화 학습을 통해 치즈를 찾아가는 모습을 나타낸 것이다. 처음에 쥐는 다양한 방향으로 시도를 할 것이고 각 단계에서 실패와 성공을 거듭할 것이다. 그 후 쥐는 여러 번의 시도(학습)을 기억하고 점점 빠르게 치즈를 찾을 것이다. 이것이 강화 학습이다. 심층 강화 학습에서 쥐는 에이전트이며 학습을 수행하는 주체이고 쥐의 뇌는 인공 신경망에 해당된다. 환경은 에이전트가 행동을 수행하는 공간으로 방과 보상으로 구성되며, 각 방은 쥐의 위치를 알려주는 상태이고 치즈는 쥐가 받아야 할 보상이다. 심층 강화 학습에서 인공 신경망은 환경과 상호작용을 통해 빠르게 치즈를 찾아가는 방법을 학습한다. 본 논문에서는 먼저 강화 학습의 이해를 위해 모든 상태와 보상을 아는 상황에서 강화 학습을 시행하는 다이나믹 프로그래밍을 기술하였으며, 심층 강화 학습의 한 종류인 A3C를 이용하여 EXAFS 데이터를 정량적으로 분석하는 예를 소개하였다.

## II. 다이나믹 프로그래밍과 시간차 학습 (Temporal Difference Learning)

### 2.1. 다이나믹 프로그래밍

다이나믹 프로그래밍[11]은 완전한 환경 모델, 즉 모든 상태와 보상을 알고 있을 때 최적의 정책이나 가치 함수를 계산하는 방법이다. 강화 학습을 이해하기 위해서는 다이나믹 프로그래밍에 대한 이해가 선행되어야 한다. 그림 1(b)와 유사한 상황을 만들기 위해 그림 2와 같이 2×3 그리드 상황을 만들었다. 이 상태에서 다이나믹 프로그래밍의 목적은 A에서 F로 이동하는 최적의 경로를 찾는 것이다. 각 상태로 이동할 때 얻는 보상은 F에 도달할 때만 1을 받는 것으로 하였으며 나머지 이동에 대해서는 보상이 0이다. 또, 각 상태에서 이동할 수 있는 방법은 4개 방향이 모두 가능하며 6개 그리드 밖으로 이동할 경우는

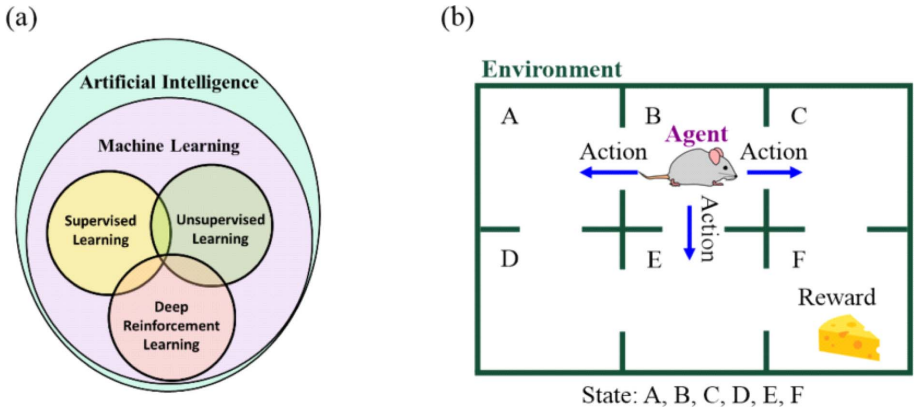
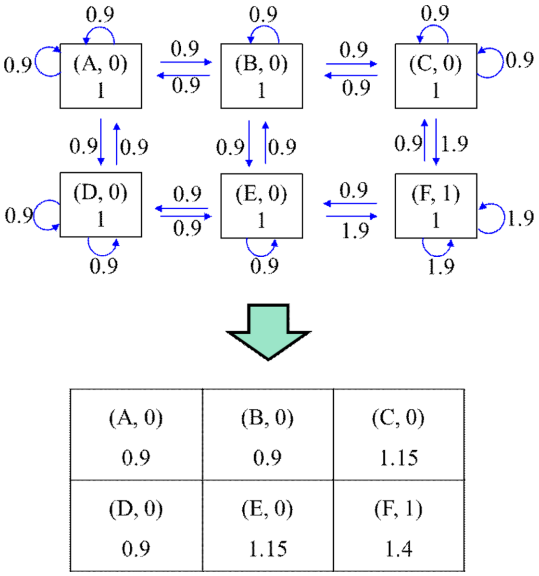



그림 1. (a) 머신러닝의 세부 분류와 (b) 실험 쥐의 강화 학습 개념 설명도




| | | |
|---|---|---|
| (A, 0)<br>0.9 | (B, 0)<br>0.9 | (C, 0)<br>1.15 |
| (D, 0)<br>0.9 | (E, 0)<br>1.15 | (F, 1)<br>1.4 |

그림 2. A에서 F로 이동하는 최적의 경로를 찾기 위한 그리드(상태, 보상). 각 상태의 초기 가치함수 값은 1로 지정, 초기 $Q$ 값은 화살표 근처의 숫자. 위는 초기 상태, 아래는 한 번 업데이트된 상태

자기 자신으로 돌아오도록 경계 조건을 갖게 하였다. A부터 F까지 이동하는 경로는 자기 자신으로 돌아오는 경우를 모두 고려하였을 때 무수히 많다. 따라서 단순히 통계적인 방법을 사용하여 A에서 F로 이동하는 경로를 찾는 것은 엄청난 노력이 요구된다. 다이나믹 프로그래밍에서 최적의 경로를 찾기 위한 방법으로는 가치 반복 학습과 정책 반복 학습이 있다. 가치 반복 학습은 가치 함수(식 (1))를 최적 정책(이동할 확률, 식 (1)의 $\pi(a|s)$)에 대한 가치 함수로 가정한 후 반복적으로 계산하여 최적의 가치를 찾아 목표에 도달하는 방법이다. 이 학습에서는 정책을 명시적으로 고려하지 않고 가치 함수를 지속적으로 업데이트하면서 원하는 목표에 도달하게 된다. 반면, 정책 반복 학습은 정책을 기반으로 목표에 더 빠르게 도달할 수 있는 행동을 강화하는 방향으로 정책을 수정하는 방법이다. 본 논문에서는 정책 반복 학습 위주로 설명을 할 것이다.

강화 학습으로 주어진 상황을 해결하기 위해서는 순차적인 의사 결정 과정으로 문제를 만들어야 한다. 순차적인 의사 결정 과정이라 함은 처음 A 상태에 있다고 가정하면 A가 바로 F로 가는 것이 아니라 단계적으로 이동하여 F에 도달하는 과정을 말한다. 각 상태에서 이동을 한 칸 또는 제자리로 돌아오는 것으로 구성하면 A~F로 이동하는 과정은 순차적인 의사 결정 과정이 된다. 순차적인 의사 결정으로 문제를 만든 후 식 (1)과 같이 벨만 방정식으로 일어나는 현상을 풀 수 있도록 보상을 정하면 특정 상태의 가치 함수는 결정된다.

$$V_\pi = \sum \pi(a|s)(r_{(s,a)} + \gamma V_\pi(s')) \quad \text{(벨만 방정식)} \qquad (1)$$

$$Q = r_{(s,a)} + \gamma V_\pi(s') \qquad (2)$$

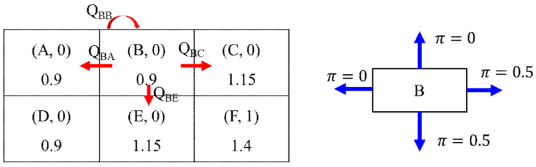
이 식에서 정책 $\pi(a|s)$는 $s$ 상태에서 어떤 행동($a$)을 취할 것인지에 대한 확률로 $s$상태에서 주변의 $s'$ 상태로 이동할 때 갖는 확률이다. $r_{(s,a)}$는 상태 $s$에서

행동($a$)을 해서 상태 $s'$로 이동할 때 받는 보상이며 $\gamma$는 감쇄 인자로 미래에 받는 보상 정도를 조절할 수 있는 상수이다. 감쇄 인자는 수렴 정도에 따라 사용자가 적절하게 조절해야 하는 값이다. $V_\pi(s')$는 정책 $\pi(a|s)$를 따라 행동했을 때 상태 $s'$에 있는 가치 함수의 값을 나타낸다. $s$ 상태에서 행동하여 이동할 수 있는 $s'$ 상태들을 위 식에 맞추어 모두 더하면 $s$ 상태의 가치 함수는 업데이트된다. 식 (2)의 $Q$-함수는 정책 $\pi(a|s)$를 따라 행동했을 때 얻는 값으로 행동의 크기를 결정할 수 있는 값이다. 그림 2에서 정책 반복 학습은 초기 정책(모든 방향에 대해 0.25)과 초기 가치 함수 값(1)을 가지고 가치 함수의 값을 업데이트하고 업데이트된 가치 함수 값으로 $Q$ 값들을 업데이트여 $Q$ 값으로부터 새롭게 정책 $\pi(a|s)$를 업데이트하는 방법이다. 이렇게 업데이트된 정책 $\pi(a|s)$는 에이전트가 어떤 상태($s$)에서 행동($a$)을 결정하는 확률로 사용된다.

그림 2에서 초기에 모든 방향에 대한 모든 정책($\pi(a|s)$)을 0.25 (동일한 확률로 지정)로 하고 감쇄 인자 ($\gamma$)를 0.9로 할 때 계산되는 $Q$ 값은 파란색 화살표 옆의 숫자와 같다. 각 화살표는 각 상태에서 이동 가능한 경로를 나타낸 것이며 그리드를 벗어나는 행동에 대해선 자기 자신으로 돌아오도록 행동을 만들었다. 식 (1)을 고려할 때, 처음 업데이트되는 가치함수를 계산하면 아래와 같다.

$$V_A(s') = 0.25(0+0.9\times(1))\times 4 = 0.9$$
$$V_B(s') = 0.25(0+0.9\times(1))\times 4 = 0.9$$
$$V_C(s') = 0.25[(0+0.9\times(1))\times 3+(1+0.9\times(1))] = 1.15$$
$$V_D(s') = 0.25(0+0.9\times(1))\times 4 = 0.9$$
$$V_E(s') = 0.25[(0+0.9\times(1))\times 3+(1+0.9\times(1))] = 1.15$$
$$V_F(s') = 0.25[(0+0.9\times(1))\times 2+(1+0.9\times(1))\times 2] = 1.4$$

새롭게 얻은 가치 함수 값들을 통해 새롭게 $Q$ 값을 업데이트하고 이를 정책에 반영시킬 수 있다. 그림 3은 새로운 가치 함수 값을 가지고 그리드 B 상태에서 $Q$ 값을 계산하는 과정을 나타낸 것이다. 그림 2의 초기 상태에서 그리드 B의 $Q$ 값들은 모든 방향에 대해 0.9로 동일하다. 즉 모든 방향에 대한 정책 $\pi(a|s)=0.25$이다. 하지만 그림 3과 같이 새로운 가치 함수에 대해 $Q_{BA} = Q_{BB} = 0 + 0.9 \times (0.9) = 0.81$, $Q_{BC} = Q_{BE} = 0 + 0.9 \times (1.15) = 1.035$이다.




각 상태에서 정책

| 상태 | 상 | 하 | 좌 | 우 |
|---|---|---|---|---|
| A | 0.25 | 0.25 | 0.25 | 0.25 |
| B | 0 | 0.5 | 0 | 0.5 |
| C | 0 | 1 | 0 | 0 |
| D | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 1 |
| F | 0 | 0.5 | 0 | 0.5 |

그림 3. 한 번 업데이트된 후 $Q$ 값과 정책

즉 $Q_{BA} = Q_{BB} < Q_{BC} = Q_{BE}$가 되어 최대 $Q$ 값은 2개가 된다. 따라서 정책은 최대 $Q$를 선택하도록 업데이트되어 $\pi(a|s)_{BC} = \pi(a|s)_{BE} = 0.5$가 된다. 같은 방법으로 계산하였을 때 각 그리드에서 정책은 그림 3의 표와 같다.

한 번 업데이트된 후 상태 A에서 이동할 확률은 0.25로 모든 방향에 대해 동일하다. 그리고 B 상태에 있을 경우 이동할 확률은 E와 C로 각각 0.5이다. 즉 A로는 가지 않는다는 것이다. 만일 A에서 출발하여 D로 이동한다면, D에서 E로 그리고 E에서 F로 갈 확률이 각각 1이 된다. 정책 반복 학습을 계속 반복하다 보면 어떤 상태에서 어떤 행동을 취해야할 지 명확하게 알 수 있다. 이러한 반복 학습이 다이나믹 프로그래밍의 정책 반복 학습이다.

## 2.2. 시간차 학습

다이나믹 프로그래밍은 모든 상태에 대해 보상 및 가치함수를 알고 있어야 한다. 간단한 상황에서는 이러한 다이나믹 프로그래밍 방법이 가능하지만, 현실적으로 모든 상태와 보상 그리고 가치 함수 값을 정확히 아는 것은 불가능에 가깝다. 설령 모든 정보를 안다고 하더라도 계산량이 엄청나게 커져 비효율적일 수 있다. 이를 극복하기 위해 시간차 학습(Temporal Difference, TD) 방법이 제시되었다[12]. 시간차 학습에서는 현재 상태에서 다른 상태로 이동할 때 현재 가치 함수의 값은 현재의 보상과 이후 상태의 가치 함수값을 바탕으로 약간 변화한 값으로 업데이트된다. 이를 수식으로 표현하면 식 (3)과 같다:

$$V(s) \leftarrow V(s) + \alpha[r_{s'} + \gamma V(s') - V(s)] \quad (3)$$

$r_{s'}$ 은 $s'$으로 이동했을 때 받는 보상, $\gamma$는 감쇄 인자, $V(s')$은 $s'$에 대한 가치 함수 값 그리고 $\alpha$는 Learning rate로 가치 함수가 업데이트되는 정도이다. 이 식은 현재 가치 함수를 시간차 에러($r_{s'} + \gamma V(s') - V(s)$, 정책 함수의 Q-함수에 해당됨)를 바탕으로 조금씩 업데이트하는 과정이다. 시간차 학습의 장점은 모든 상태와 보상을 미리 알 필요가 없으며, 경험을 통해 점진적으로 가치함수를 업데이트할 수 있다는 점이다. 이로 인해 계산 효율성이 크게 향상되며, 현실적인 환경에서도 적용이 가능해진다.

## III. 심층 강화 학습 (Deep Reinforcement Learning)

심층 강화 학습은 인공 신경망을 결합한 강화 학습으로 DQN (Deep Q-Network), DDPC (Deterministic Policy Gradient), PPO (Proximal Policy Optimization), A3C (Asynchronous Advantage Actor Critic) 등과 같이 다양한 알고리즘이 있다[13-16]. 본 논문에서는 구조적으로 이해하기 쉬운 A3C 강화 학습 알고리즘을 사용하여 EXAFS 분석에 사용된 예를 설명하면서 심층 강화 학습의 응용 사례를 보여주고자 한다. 그림 4와 같이 A3C는 정책 신경망과 가치 신경망을 포함하는 구조를 가지고 있다. 정책 신경망과 가치 신경망은 다이나믹 프로그래밍의 정책 반복 학습에서 제시된 정책 함수와 가치 함수를 만들어내는 역할을 한다. 각 상태가 입력으로 주어졌을 때, 정책 신경망은 행동에 대한 확률 분포(정책)를 출력하며 가치 신경망은 해당 상태의 가치 함수를 출력한다. 행동 확률은 정책 신경망을 평가하고 업데이트하기 위해 크로스 엔트로피 함수로 입력되고 새로운 가치 함수는 식 (3)에 의해 시간차 오차를 계산할 수 있도록 입력된다. 시간차 오차는 오차 역전파를 통해 가치 신경망의 업데이트와 정책 신경망의 업데이트를 위해 사용된다. 그림 4와 같이 크로스 엔트로피 함수와 시간차 오차와의 곱으로 손실 함수를 계산해서 오차 역전파로 정책 신경망은 업데이트된다. 이러한 구조적 특성을 통해 A3C는 정책과 가치 신경망 학습을 실시간 병렬적으로 수행할 수 있어 기존 강화 학습 기법보다 효율적이고 빠르게 수렴할 수 있다.

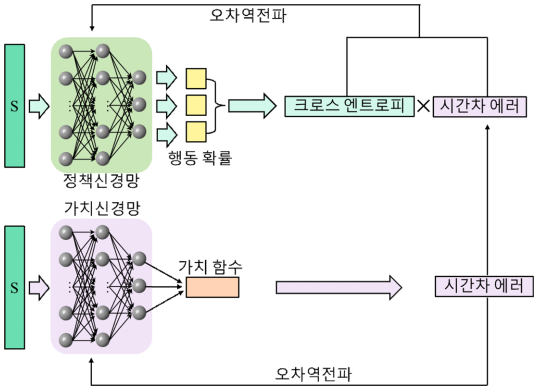



그림 4. A3C 심층 강화 학습 개념도

EXAFS 데이터 분석에서 입력되는 상태 $s$는 이론적인 EXAFS 신호이며, 각각 정책 및 가치 신경망으로 들어간다. 행동 확률은 데이터 맞추어보기 (fitting)에서 얻는 변수의 확률로 변수의 값을 결정한다. 예를 들면 하나의 광전자의 경로에 대해 찾아야 할 변수는 흡수 문턱 에너지 $\Delta E$, 중심 원자로부터 같은 거리에 존재하는 이웃 원자들의 수 $N$, 중심 원자에서 이웃 원자 간의 거리 $\Delta R$, 중심 원자와 이웃 원자 간의 거리 무질서도 $\sigma^2$ 이다. $\Delta E$, $N$, $\Delta R$, $\sigma^2$ 중 최적의 $\Delta E$를 찾는 경우를 고려하면, 정책 신경망을 통해 보

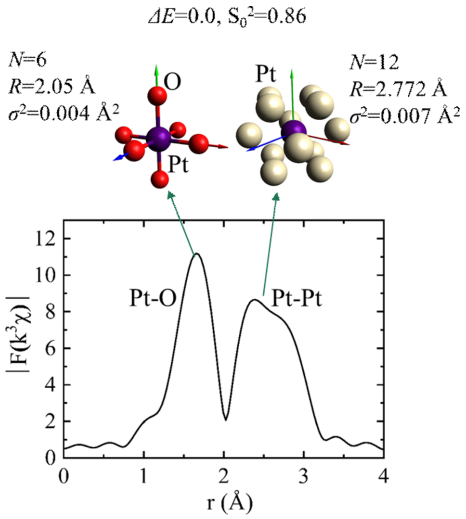
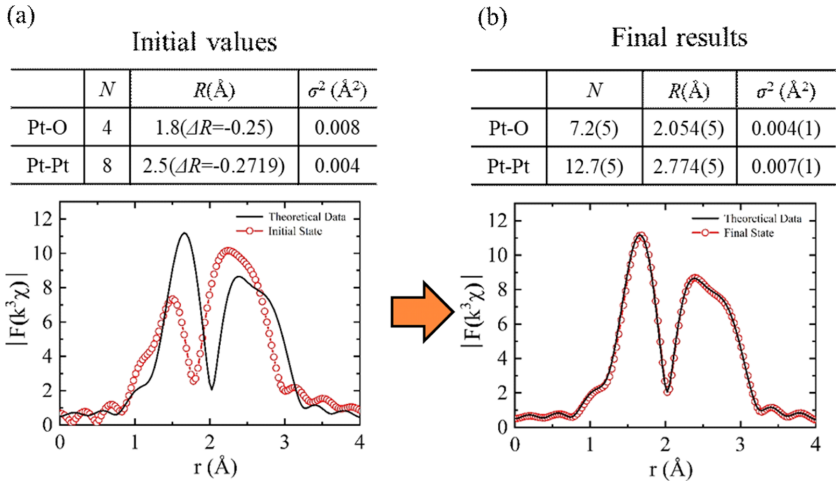



그림 5. Pt 주위에 6개의 산소와 12개의 Pt 원자가 있는 구조로 가정하고 이론적으로 계산한 EXAFS(r), $|F(k^3\chi)|$




| | N | R(Å) | σ² (Å²) |
|---|---|---|---|
| Pt-O | 4 | 1.8(⊿R=-0.25) | 0.008 |
| Pt-Pt | 8 | 2.5(⊿R=-0.2719) | 0.004 |

| | N | R(Å) | σ² (Å²) |
|---|---|---|---|
| Pt-O | 7.2(5) | 2.054(5) | 0.004(1) |
| Pt-Pt | 12.7(5) | 2.774(5) | 0.007(1) |

그림 6. A3C를 이용한 EXAFS(r)의 데이터 맞추어보기 결과. (a) 초기 입력된 변수들의 값, EXAFS, (b) A3C로 찾아낸 결과와 마지막 결과 상태

상을 최대로 받을 $\Delta E$를 행동 확률로 찾는 것이 심층 강화 학습의 목표이다. 반복적으로 찾는 과정이 진행되면서 정책 신경망과 가치 신경망이 업데이트되고, 최적의 행동을 갖는 변수들이 결정된다.

A3C 알고리즘에서는 정책 신경망이 각 변수의 값을 행동으로 선택하고 이를 환경에 전달하면서 지속적으로 학습을 진행한다. 이 과정에서 정책 신경망과 가치 신경망은 점진적으로 발전하면서 최적의 행동, 즉 최적의 변수를 결정할 수 있도록 업데이트된다. 본 논문에서는 이론적으로 계산된 $PtO_x$ EXAFS 데이터를 A3C를 이용한 맞추어보기 (fitting)를 통하여 분석하였다. 그림 5는 FEFF [17,18]를 이용하여 이론적으로 계산된 EXAFS를 보여준다. 이론 EXAFS 계산에서 중심 원자는 Pt이며, 주변 원자 O와 Pt에 대해 Pt-O 원자 쌍의 배위수 $N$ = 6, 거리 $R$ = 2.05 Å, 거리의 무질서도 $\sigma^2$ = 0.004 Å이고, Pt-Pt 원자 쌍의 배위수 $N$ = 12, 거리 $R$ = 2.772 Å, 거리의 무질서도 $\sigma^2$ = 0.007 Å으로 지정하였다. 흡수 문턱 에너지 변화는 $\Delta E_0$ = 0.0eV이고, $S_0^2$ = 0.86 으로 하였다. 주어진 이 값들은 임의의 값으로 시작하는 신경망이 최종적으로 찾아야 하는 값들이다.

그림 6은 신경망에 주어진 EXAFS 데이터(검정색)와 변수들에게 준 초기 입력 값을 이용하여 계산된 EXAFS (빨간색)를 보여준다. 변수들에게 준 초기 입력값은 위에서 이론 EXAFS를 계산할 때 사용된 값과 상당히 다르므로 EXAFS 데이터와 상당한 차이가 있음을 알 수 있다. A3C는 반복적인 시도와 학습을 통하여 최종적으로 그림 6(b)와 같은 결과를 얻는다. 신경망이 찾은 최종 결과는 오차 범위 내에서 정답 값들과 유사하다는 것을 확인할 수 있다. 훈련된 A3C 신경망을 사용할 경우 초기 값을 가지고 시작하여 맞추어보기를 완성하는 데까지 걸리는 시간은 약 2분 정도 소요된다. 초기 상태의 데이터는 실제 데이터와 매우 다른 상태에서 시작했음에도 불구하고 맞추어보기가 완성된 상태의 결과는 실제 데이터 결과와 상당히 잘 맞는다는 것을 알 수 있다.

## IV. 결 론

본 연구는 다이나믹 프로그래밍을 사용하여 2×3 그리드 환경에서 최적의 이동 경로를 찾는 방법을 정책 반복 학습으로 보여주었다. 정책 반복 학습을 통해 에이전트의 행동 확률(정책)이 점진적으로 발전하며, 이를 통해 최단 경로를 정확하게 예측할 수 있었다. 정책 반복 학습 방법과 시간차 오차 학습 기법을 심층 강화 학습에 적용한 알고리즘 중 A3C를 선택하고 A3C의 구조와 작동에 대해 논하였다. A3C 심층 강화 학습은 복잡한 데이터 분석인 EXAFS 분석에서 효과적으로 사용될 수 있음을 이론적인 $PtO_x$ EXAFS 데이터의 정량적인 분석을 통해 확인하

였다. 이것은 심층 강화 학습이 과학 실험 데이터를 정량적으로 분석하는데 효과적으로 활용될 수 있음을 암시하는 것이다. 심층 강화 학습은 주어진 환경과 상황에 따라 유연하게 대처할 수 있는 특성을 지니고 있어 다양한 시스템이나 문제 상황에서도 적용 가능하다. 본 연구는 심층 강화 학습 방식이 과학 분야에서 정밀한 데이터의 정량적인 분석과 새로운 연구 기법의 개발에 있어 중요한 도구로 활용될 것으로 기대된다.

## 참고문헌

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Nat., **323**, 9 (1986).
[2] J. Jumper et. al, Nat., **596**, 582 (2021).
[3] M. M. Ahsan, S. A. Luna, and Z. Siddique, Healthcare, **10**, 541 (2022).
[4] P. Courtiol, C. Maussion, M. Moarii, E. Pronier, S. Pilcer, M. Sefta, P. Manceron, S. Toldo, M. Zaslavskiy, N. Le Stang, N. Girard, O. Elemento, A. G. Nicholson, J.-Y. Blay, F. Galateau-Salle, G. Wainrib, and T. Clozel, , Nat. Med., **25**, 1519 (2019).
[5] A. Ali, S. A. Razak, S. H. Othman, T. A. E. Eisa, A. Al-Dhaqm, M. Nasser, T. Elhassan, H. Elshafie, and A. Saif, Appl. Sci., **12**, 9637 (2022).
[6] J. Hou and A. Zhu, Appl. Sci., **13**, 5321 (2023).
[7] S. K. Sahu, A. Mokhade, and N. D. Bokde, Appl. Sci., **13**, 1956 (2023).
[8] F.-L. Vincent, H. Peter, I. Riashat, G. B. Marc, and P. Joelle, Found. Trends Mach. Learn., **11**, 219 (2018).
[9] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, Nat., **549**, 196 (2017)
[10] J. Wei, X. Chu, X. -Y. Sun, K. Xu, H. -X. Deng, J. Chen, Z. Wei, and M. Lei, InfoMat., **1**, 338 (2019).
[11] D. J. White and J. Math. Anal. Appl., **6**, 373 (1963).
[12] G. Tesauro, Machine Learning, **8**, 257 (1992).
[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, L. Antonoglou, D. Wierstra, and M. Riedmiller, **1** (2013). arXiv:1312.5602[cs.LG]
[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, **6** (2019). arXiv:1509.02971 [cs.LG]
[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, **2** (2017). arXiv:1707.06347 [cs.LG]
[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, **2** (2016). arXiv:1602.01783 [cs.LG]
[17] J.J. Rehr and R.C. Albers, Rev. Mod. Phys., **72**, 621 (2000).
[18] J.J. Rehr, J.J. Kas, F.D. Vila, M.P. Prange, and K. Jorissen, Phys. Chem. Chem. Phys., **12**, 5503 (2010).

## 다이나믹 프로그래밍 정책 반복 학습의 파이썬 코드

```
import numpy as np

class Environment:
    """그리드월드 환경 정의"""
    def __init__(self):
        self.grid_size = (2, 3)  # 2행 3열
        self.rewards = np.array([
            [0, 0,  0],  # 첫 번째 행
            [0,  0,  1]   # 두 번째 행
        ])  # 보상 정의
        self.terminal_states = [(1, 2)]  # 종료 상태

    def get_next_states(self, state):
        row, col = state
        rows, cols = self.grid_size
        # 상하좌우 이동 정의
        actions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
        next_states = []
        for dr, dc in actions:
            next_row, next_col = row + dr, col + dc
            # 경계를 벗어나지 않도록 제약
            if 0 <= next_row < rows and 0 <= next_col < cols:
                next_states.append((next_row, next_col))  # 정상적으로 이동
            else:
                next_states.append((row, col))  # 경계를 벗어나면 자기 자신으로 돌아옴
        return next_states
class Agent:
    def __init__(self, environment):
        self.env = environment
        self.gamma = 0.9
        self.policy = np.full((*self.env.grid_size, 4), 0.25)  # 초기 행동 확률

    def policy_evaluation(self, value_function):
        new_value_function = np.copy(value_function)
        for row in range(self.env.grid_size[0]):
            for col in range(self.env.grid_size[1]):
                state = (row, col)
                # 새로운 상태의 가치 계산
                state_value = 0
                next_states = self.env.get_next_states(state)

                for i, next_state in enumerate(next_states):
                    next_row, next_col = next_state
                    reward = self.env.rewards[next_row, next_col]
```

```
                    state_value += self.policy[row, col, i] * (
                        reward + self.gamma * value_function[next_row, next_col] )
                new_value_function[row, col] = state_value
        return new_value_function

    def policy_improvement(self, value_function):
        for row in range(self.env.grid_size[0]):
            for col in range(self.env.grid_size[1]):
                state = (row, col)
                next_states = self.env.get_next_states(state)
                Q_values = np.zeros(4)
                for i, next_state in enumerate(next_states):
                    next_row, next_col = next_state
                    reward = self.env.rewards[next_row, next_col]
                    Q_values[i] = reward + self.gamma * value_function[next_row,
next_col]
                max_idx_list = np.argwhere(Q_values == np.amax(Q_values)).flatten().tolist()
                prob = 1 / len(max_idx_list)
                new_policy = np.zeros(4)
                for idx in max_idx_list:
                    new_policy[idx] = prob
                # 정책 업데이트
                self.policy[row, col] = new_policy
def main():
    env = Environment()
    agent = Agent(env)

    max_iterations = 1
    initial_value_function = np.ones(env.grid_size)  # 가치 함수 초기화
    value_function=initial_value_function
    new_value_function=initial_value_function

    for i in range (max_iterations):
        new_value_function=agent.policy_evaluation(value_function)
        agent.policy_improvement(value_function)
        value_function=new_value_function
    print("₩n최적 가치 함수:")
    print(new_value_function)
    print("₩n최적 정책:")
    for row in range(env.grid_size[0]):
        for col in range(env.grid_size[1]):
            print(f"상태 ({row}, {col}): {agent.policy[row, col]}")

if __name__ == "__main__":
    main()
```