학습과 행동을 분리한 분산형 DQN 모델

홍기현*, 임준식**
*가천대학교 IT융합공학과
**가천대학교 IT융합대학 컴퓨터공학과
e-mail : *ghdrlgus96@naver.com, **jslim@gachon.ac.kr

Decentralized DQN Model that Separates Learning and Behavior

*Ki Hyeon Hong, **Jun-S. Lim
*Dept. of IT Convergence Engineering, Gachon University
**Dept. of Computer Engineering, College of IT, Gachon University

요 약

인공지능 분야에서 강화학습은 에이전트가 외부 환경과 상호작용하면서 목표를 달성하는 목표 지향적 기계학습 방법이다. 강화학습 알고리즘 중 하나인 DQN 모델은 에이전트의 상태와 행동에 따른 보상의 기댓값을 학습에 사용하는 방법이며, 아타리 2600의 게임 중 29개 게임에서 인간을 뛰어넘는 기록을 세웠다. 그러나 기존의 DQN 모델은 리플레이 메모리에 에이전트의 행동으로 얻은 샘플을 수집하고, 이를 학습에 이용한다는 점에서 리플레이 메모리의 큰 용량에 대한 부담과 과거의 데이터를 학습하는 한계가 있다. 따라서 본 논문에서는 리플레이 메모리를 이용하여 학습하는 과정을 글로벌 네트워크로, 에이전트가 외부 환경과 상호작용하여 얻은 샘플을 리플레이 메모리에 전송하는 과정을 러너로 분리하고 러너를 다중 생성하여 샘플을 좀 더 빠르게 수집하는 개선된 DQN 모델을 제안한다. 또한, 제안한 DQN 모델의 실용성을 확인하기 위해 실제 BreakOut 게임을 학습하는 실험을 하였으며, 그 결과 기존의 DQN 모델에 비해 학습률과 학습 속도 면에서 개선됨을 확인할 수 있었다. 결론적으로 본 논문에서 제안한 학습과 행동 분산형 DQN 모델은 글로벌 네트워크와 러너로 분리될 수 있는 환경에서 빠르고 높은 성능으로 학습을 수행하는 데에 기여할 것이다.

1. 서론

강화학습은 기계학습의 한 분야로 에이전트가 외부 환경과의 상호작용을 통해 목표를 이루는 방향으로 학습하는 방법이다[1]. 그러나 외부 환경은 무수한 다양한 상태와 변수가 존재하며, 에이전트는 매우 다양한 행동을 선택할 수 있다. 따라서 이를 전부 기억하고 행동하는 것은 사실상 불가능하다.

이러한 문제를 해결하기 위해 상태와 행동에 따른 보상을 인공신경망으로 근사하는 방법이 계속하여 연구되고 있다[2]. 특히, 2015년 구글의 딥마인드에서는 아타리 2600의 게임 중 29개 게임에서 인간을 뛰어넘는 기록을 세운 DQN 모델을 제시하였으며, 이를 통하여 강화학습이 인간을 뛰어넘을 수 있음을 보여주었다[2].

그러나 기존의 DQN 모델은 학습을 위해 샘플을 저장하는 리플레이 메모리가 필요하다. 이는 메모리가 상주할 공간의 문제와 학습과 행동을 하나의 에이전트에서 수행한다는 점 때문에 에이전트의 연산 능력이 매우 중요하다. 이러한 점은 저사양 IoT와 같은 환경에 쉽게 적용할 수 없다는 한계가 있다. 또한, 리플레이 메모리의 샘플을 학습에 이용한다는 점에서 과거의 데이터를 학습에 이용하므로 학습을 방해하는 요소로 작용할 수 있다[3]. 이를 해결하기 위해서는 리플레이 메모리의 데이터를 비교적 최신의 데이터로 유지하며, 리플레이 메모리가 상주하는 환경을 분리하여 메모리 공간에 대한 부담을 고려한 개선된 DQN 모델이 필요하다.

따라서 본 논문에서는 학습과 행동을 분리한 분산형 DQN 모델을 제안한다. 이 모델은 글로벌 네트워크와 러너로 구성된다. 글로벌 네트워크는 러너로부터 받은 샘플을 저장하는 리플레이 메모리, 글로벌 타겟 모델, 그리고 글로벌 학습 모델로 구성되며, 리플레이 메모리의 샘플을 사용하여 글로벌 학습 모델과 글로벌 타겟 모델의 가중치를 업데이트한다. 러너는 업데이트된 글로벌 학습 모델의 가중치를 자신의 로컬 모델의 가중치로 업데이트한 후, 외부 환경과 상호작용을 통하여 새로운 샘플을 수집한다. 이때, 러너는 비동기적으로 다수 존재하며, 각각의 러너는 각기 다른 환경에서 다양한 데이터를 수집한다. 이를 통하여 러너는 리플레이 메모리 공간에 대한 부담을 받지 않으며, 글로벌 네트워크는 많은 러너로부터 데이터를 수집하여 리플레이 메모리를 갱신하므로 과거의 데이터를 학습에 사용하는 단점을 개선한다. 이를 기존의 DQN 모델과 본 논문에서 제안한 DQN 모델의 학습 속도 및 학습률을 비교하는 실험을 통하여 확인한다.

본 논문의 2장에서는 강화학습과 기존의 DQN 모델의 한계점을 고찰하고 이를 해결하기 위한 기존의 연구를 살펴본다. 3장에서는 학습과 행동 분산형 DQN 모델을 정의하며 모델을 구성하는 글로벌 네트워크와 러너를 구현하는 알고리즘을 제시한다. 4장에서는 본 논문에서 제안한 DQN 모델의 실용성을 확인하기 위한 실험을 설계 및 수행하고 결과를 분석한다. 마지막 5장에서는 결론을 맺는다.

2. 관련 연구

2.1 강화학습

강화학습에서 에이전트가 외부 환경과 상호작용하며 학습하는 과정은 그림 1.과 같다. 그림 1.의 에이전트는 현재 상태에서 행동(a)를 결정하며 환경으로부터 다음 상태(s)와 그에 대한 보상(r)을 받는다. 이를 통하여 에이전트는 어느 행동이 좋은지를 학습한다. 이것이 에이전트와 환경의 상호작용이며, 에이전트는 상태, 행동, 그리고 보상을 통하여 가장 많은 보상을 받는 쪽으로 학습을 진행한다. 이때, 특정 상태에서 어떠한 행동이 보상을 받을 기댓값이 높은지 반환하는 함수가 큐함수이며, 에이전트는 큐함수와 환경으로부터 받은 보상의 오차를 줄이는 것으로 큐함수를 업데이트한다[4].

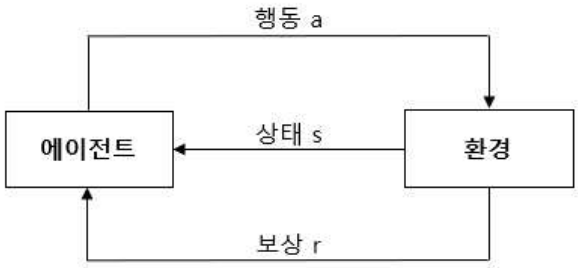

그림 1. 강화학습에서의 에이전트와 환경의 상호작용

그러나 현실 세계의 환경은 매우 복잡하고 변수가 많으며, 에이전트는 이러한 환경에서 매우 다양한 상태와 행동을 갖는다. 따라서 큐함수가 갖는 모든 입력과 출력 쌍을 저장하는 것은 매우 비효율적이다. 이를 해결하기 위해 큐함수의 경향을 파악하고, 근사하게 되는데 이를 위해 사용할 수 있는 것이 바로 인공신경망이다[5].

인공신경망으로 큐함수를 근사하기 위해 오차를 최소화시킨다. 이때, 오차는 목표의 실제 값과 큐함수 예측의 차이이다. 그러나 큐함수가 근사하고자 하는 목표는 목표 지점에 도달할 때까지의 보상의 총합이며, 이는 하나의 에피소드가 끝난 후에야 알 수 있다. 이러한 점에서 기존의 인공신경망으로 큐함수를 근사하는 방법은 에피소드가 매우 긴 작업이나, 종료가 없는 작업에서 활용하기 어려우며, 이를 해결하기 위해 목표의 실제 값을 다른 값으로 대체해야한다[2].

2.2 DQN 알고리즘과 한계점

구글의 딥마인드에서는 DQN이라는 강화학습 알고리즘을 소개하였으며, 이는 타겟 큐함수라는 목표의 실제 값의 역할을 하는 큐함수를 두어 이를 해결하였다[6]. 그림 2.의 타겟 큐함수는 목표 지점에 도달할 때까지 앞으로 받을 보상의 총합을 반환하는 역할을 하며, 이를 위해 다음 상태(s')를 입력으로 받아 결과의 최댓값을 취한다. 만약 타겟 큐함수를 두지 않고 큐함수만으로 정답과 예측을 전부 수행한다면 매 에피소드마다 업데이트되는 큐함수의 학습 과정에서 불안정하므로 일정 시간 고정된 타겟 큐함수로 목표값을 예측한다[6].

또한, 에이전트는 환경과 상호작용을 하면서 데이터를 수집하기 때문에 현재 학습할 데이터는 이전의 데이터, 다음의 데이터와 높은 연관성을 갖는다. 이는 에이전트가 학습한 행동이 하나로 치우쳐지는 경향을 보일 수 있다, 예를 들어 BreakOut 게임에서 우연히 왼쪽으로 이동하는 것이 점수를 높게 받는다는 것을 에이전트가 학습하게 된다면 에이전트는 더 이상 오른쪽으로는 이동하지 않게 될 가능성이 높다. 이를 해결하기 위해 DQN 모델에서는 리플레이 메모리를 사용한다[6].

리플레이 메모리는 에이전트가 얻은 샘플을 저장하는 메모리이며, 학습을 위해 리플레이 메모리의 데이터를 랜덤으로 뽑아 학습에 사용한다. 이를 통하여 현재 학습할 데이터는 이전에 학습한 데이터, 다음에 학습할 데이터와의 연관성이 낮아진다. 이는 그림 2.와 같이 동작한다. 리플레이 메모리로부터 얻은 상태, 행동으로 큐함수의 값을 예측하며, 타겟 큐함수로 다음 상태에서 얻을 보상의 최댓값을 목표값으로 한다. 이를 통하여 오차를 구하며, 이는 큐함수의 오차를 줄이는 방향으로 학습이 진행된다.

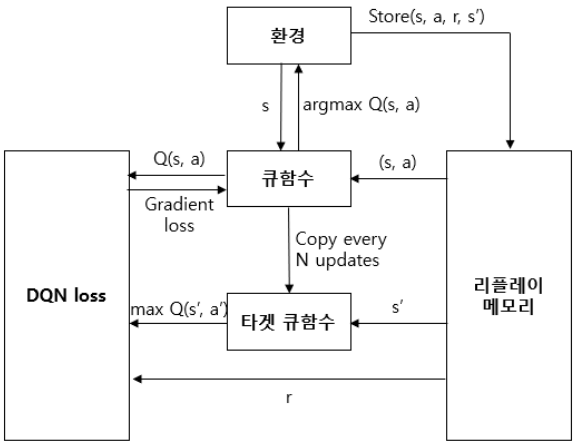

그림 2. DQN 모델의 학습 프로세스

그러나 기존의 DQN 모델이 리플레이 메모리에 샘플을 수집하고, 학습에 사용한다는 점에서 한계가 있다[7]. 예를 들어 기존의 연구에서는 아타리 2600의 게임을 학습하기 위해 100,000의 메모리 공간을 사용하였다[6]. 이는 리플레이 메모리에 샘플을 저장하고 랜덤으로 샘플을 뽑는 과정

에서 큰 비용이 발생한다. 또한, 리플레이 메모리의 데이터는 과거로부터 계속 쌓아온 샘플이며, 이는 큐함수가 업데이트되기 전에 수집한 샘플을 다수 포함한다. 이러한 점에서 큐함수를 업데이트하기 위해 과거의 데이터를 학습에 이용할 가능성이 높으며, 이는 학습을 방해하는 요소로 작용할 수 있다[7].

2.3 연합 학습(Federated Learning)

연합 학습은 각각의 고유한 로컬 및 데이터를 보유하고 있는 여러 분산 장치 또는 서버에서 기계학습 알고리즘을 훈련하는 것을 목표로 한다[8]. 이를 통하여 데이터 수집 과정에서 발생하는 개인 정보 보호 문제를 해결할 수 있는 장점이 존재한다. 또한, 연합 학습은 로컬에서 학습한 모델을 서버에 업로드하며, 업로드된 모델의 가중치를 다른 클라이언트가 공유하여 자신의 샘플로 가중치를 업데이트한다는 점에서 학습 샘플 간의 연관성을 크게 낮출 수 있다[7]. 그중 하나의 예로 A3C 모델은 이러한 연합 학습의 장점을 이용하여 데이터의 연관성을 줄였다[9].

그러나 기존의 연합 학습은 로컬에서 수집한 샘플로 모델을 학습하고, 이를 서버에 전송한다는 점에서 학습과 행동을 전부 로컬에서 수행한다고 볼 수 있다. 이는 로컬 환경의 입장에서 큰 비용이 발생하며, 로컬이 저사양 IoT 환경이라면 이러한 학습 방식을 사용하기 어렵다.

이러한 점에서 본 논문에서는 비용이 큰 학습을 서버에서 수행하고 학습된 가중치를 로컬 환경으로 받아와 업데이트된 모델로 행동을 하는 학습과 행동을 분리한 분산형 DQN 모델을 제안한다. 특히, 행동을 하는 로컬은 다수로 생성하여 빠른 샘플 수집으로 과거의 데이터를 학습에 이용한다는 기존의 DQN 모델의 한계점을 개선한다.

3. 학습과 행동 분산형 DQN 모델

3.1 학습과 행동 분산형 DQN 모델 설계

제안한 학습과 행동 분산형 DQN 모델은 리플레이 메모리의 샘플을 통해 큐함수를 업데이트하는 학습 과정과 에이전트가 큐함수의 결과값으로 행동을 수행하여 새로운 샘플을 수집하는 행동 과정을 분리한 모델이다. 이는 그림 3.과 같이 2개의 과정으로 나눈다.

그림 3.의 글로벌 네트워크는 글로벌 타겟 모델, 글로벌 학습 모델, 그리고 리플레이 메모리로 이루어져 있으며, 리플레이 메모리의 샘플을 통하여 글로벌 학습 모델의 가중치를 업데이트한다. 글로벌 타겟 모델은 가중치 업데이트를 위한 목표의 역할을 수행하며 일정 시간마다 글로벌 학습 모델의 가중치로 자신을 업데이트한다.

그림 3.의 러너는 글로벌 학습 모델의 가중치로 로컬 모델을 업데이트하며 환경으로부터 샘플을 수집하여 글로벌 네트워크의 리플레이 메모리로 샘플을 전송한다. 러너는 일정 시간마다 글로벌 네트워크의 글로벌 학습 모델의 가중치를 요청하여 로컬 모델을 업데이트한다. 러너는 여러 개가 존재할 수 있으며, 각각의 환경에서 다양한 샘플을 수집한다.

이를 통하여 글로벌 네트워크에서는 리플레이 메모리를 이용한 학습만을 수행하며, 실제 환경으로부터 샘플을 수집하는 행동은 러너로 분리된다. 러너는 리플레이 메모리의 공간 문제와 리플레이 메모리부터 샘플을 뽑는 고비용의 연산이 발생하지 않아 저사양의 IoT 로컬 환경에서 동작 가능하다.

3.2 글로벌 네트워크 구현 알고리즘

글로벌 타겟 모델, 글로벌 학습 모델, 그리고 로컬 모델의 가중치를 공유하기 위해 큐함수 클래스를 선언한다. 또

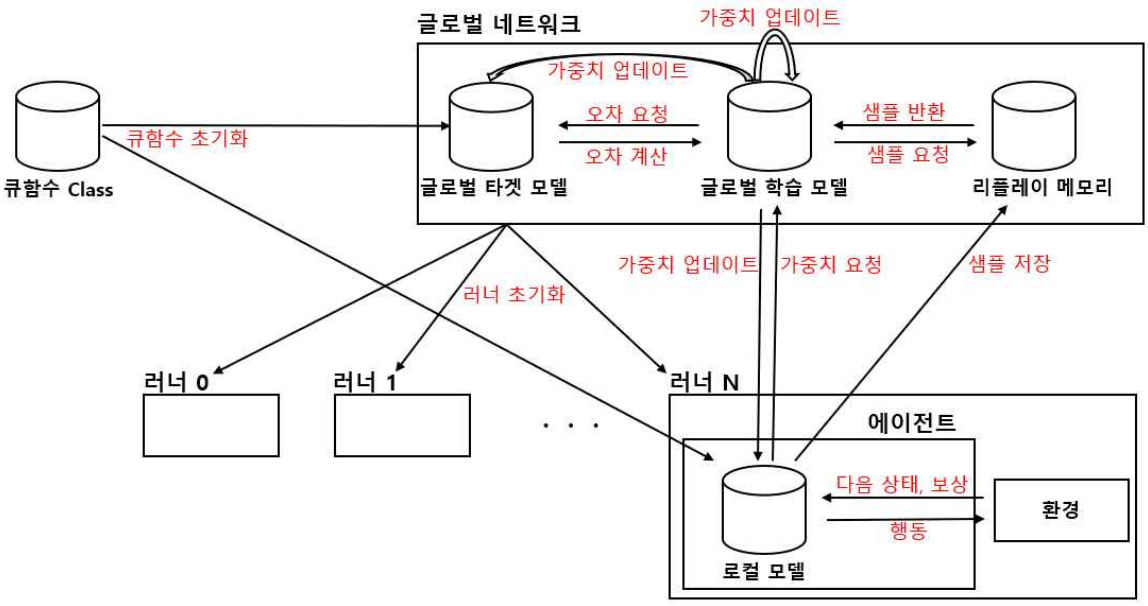

그림 3. 학습과 행동 분산형 DQN 모델

한, 글로벌 네트워크는 글로벌 타겟 모델, 글로벌 학습 모델, 그리고 리플레이 메모리를 선언해야 하며, 구현을 위한 코드는 다음 그림 4.와 같다.

글로벌 네트워크 Pseudo code

```
1   Class QFunction:
2       def __init__:
3       def call:
4
5   Class GlobalNetwork
6       global_target_model = QFunction()
7       global_train_model = QFunction()
8       global_replay_memory = deque()
9
10      def global_model_update:
11          global_target_model = global_train_model
12
13      def train:
14          runner = Runner():
15          runner.start()
16
17          while True:
18              self.train_model()
19              self.global_model_update()
20
21      def train_model:
22          batch = random.sample(global_replay_model)
23          model_params = global_train_model.variables
24          predicts = global_train_model(batch)
25          target_predicts = global_target_model(batch)
26
27          loss = sqrt(target_predicts  -  predicts)
28
29          grades = gradient(loss, model_params)
30          self.optimizer(grads, model_params)
```

그림 4. 글로벌 네트워크 pseudo code

그림 4.의 글로벌 네트워크 클래스를 보면 각각의 큐함수 모델은 큐함수 클래스를 통하여 생성하며 리플레이 메모리는 덱의 형태로 선언하여 랜덤 배치만큼 학습 샘플을 뽑을 수 있도록 한다. 그림 4.의 14번째 줄을 보면 샘플을 수집할 러너를 생성하고, 일정 시간마다 러너들이 수집한 리플레이 메모리의 샘플을 뽑아 글로벌 학습 모델이 가중치를 업데이트한다. 학습은 글로벌 타겟 모델이 예측한 값과 글로벌 학습 모델이 예측한 값의 오차만큼 가중치를 업데이트하며, 일정 주기마다 타겟 모델의 가중치를 학습 모델의 가중치로 업데이트한다.

3.3 러너 구현 알고리즘

글로벌 네트워크로부터 생성된 러너는 에이전트와 환경을 생성하며 에피소드를 수행한다. 이를 구현하기 위한 코드는 그림 5.와 같다.

그림 5.의 러너 클래스는 생성한 에이전트와 환경을 통해 샘플을 수집한다. 하나의 에피소드를 수행하며 에이전트는 환경으로 행동을 요청하고, 환경은 상태와 행동에 따른 변화된 상태와 보상을 반환한다. 이는 하나의 샘플로 리플레이 메모리로 전송되며, 하나의 에피소드가 끝난 후 글로벌 네트워크의 학습 모델의 가중치를 받아와 자신을 업데이트한다. 에이전트 클래스는 행동에 대한 기댓값을 예측하기 위해 로컬 모델을 생성하며 이를 기반으로 그림 5.의 23번째 줄과 같이 각 상태에서의 최대 기댓값을 자신이 수행할 행동으로 반환한다.

러너 Pseudo code

```
1   Class Runner:
2       def start:
3           env = Environment()
4           agent = Agent()
5
6           episode = 5000
7           for e in range(episode):
8               while not done:
9                   action = agent.get_action(state)
10                  state, reward, done = env.step(action)
11
12                  agent.append_sample(action, state, reward)
13                  agent.update_model()
14
15  Class Agent:
16      def __init__:
17          self.lcal_model = QFunction()
18
19      def update_model:
20          self.local_model = global_train_model
21
22      def get_action(state):
23          return max(local_model(state))
24
25      def append_sample(action, state, reward):
26          global_replay_memory.add(action, state, reward)
```

그림 5. 러너 pseudo code

이와 같이 제안한 DQN 모델의 실용성을 확인하기 위해 실제로 이를 구현하고 기존의 DQN 모델과 비교하고 평가하는 성능 실험을 하고자 한다. 학습률은 일정 에피소드를 진행하였을 때의 최대 큐함수 값과 최대 점수를 통해 확인하며, 학습 시간은 일정 시간에서 진행한 에피소드 수를 비교한다.

4. 실험 및 평가

4.1 실험 설계

본 실험에서는 아타리 2600 게임 중 하나인 BreakOut을 기존의 DQN 모델과 제안한 모델로 학습을 진행하였을 때 학습률과 학습 시간을 비교하고자 한다.

4.2 실험 환경

실험을 위해 기존의 DQN 모델과 제안한 모델은 Python 3.6.10 버전, Tensorflow 2.4.1 버전, 그리고 Keras 2.4.3 버전으로 구현하였다. 또한, CPU는 Intel Core i7-10700을 사용하였으며, Memory는 16GB, 그리고 Windows 10 환경에서 구현하였다. 개발 코드는 Github에서 확인할 수 있다[10].

그림 6.은 BreakOut의 게임 화면이다. 그림 6.의 (a)는 현재 게임 상태를 나타낸다. 특히, 첫 번째의 숫자는 블록이 깨질 때마다 증가하는 현재까지의 보상을 의미하며, 두 번째 숫자는 목숨을 의미한다. 하나의 에피소드는 5개의

목숨으로 구성되어 있으며 모든 목숨을 소진할 시 게임이 종료된다. 그림 6.의 (b)는 현재 게임 진행 화면이다. 게임은 공이 밑으로 떨어지지 않게 막대를 좌우로 조종하며, 공이 벽돌과 충돌할 때마다 벽돌이 사라지고 그림 6.의 (a)의 보상을 획득한다.

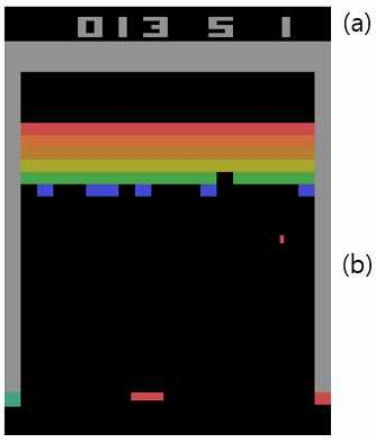

그림 6. BreakOut 게임 화면

4.3 실험결과 및 성능평가

학습과 행동 분산형 DQN 모델의 성능 실험은 본 논문에서 제안한 개선된 DQN 모델이 기존의 DQN 모델보다 학습률과 학습 시간에서 개선이 있었는지 확인하였다.

표 1.은 4,000 에피소드 동안 학습한 기존의 DQN 모델, 러너를 1개로 설정한 분산형 DQN 모델, 그리고 러너를 2개로 설정한 분산형 DQN 모델의 최대 큐함수 값과 점수 결과이다. 일정 에피소드 수행에 있어서 기존의 DQN 모델과 러너를 1개로 한 분산형 DQN 모델은 비슷한 속도의 샘플을 수집하였으며, 거의 같은 수준의 학습률을 보였다. 그러나, 제안한 러너를 1개로 한 분산형 DQN 모델은 4,000 에피소드를 수행하는 데에 있어서 기존의 DQN 모델보다 빠른 속도를 보였다. 이는 일정 시간에서 더 많은 에피소드를 진행할 수 있음을 의미하며, 학습 속도에서 개선되었음을 알 수 있었다.

표 1. 4,000 에피소드 수행에 따른 DQN 모델의 성능평가 결과

모델	최대 큐함수 값	최대 점수
기존의 DQN 모델	5.18	384
분산형 DQN 모델(러너 1)	5.15	417
분산형 DQN 모델(러너 2)	6.83	429

또한, 러너를 2개로 한 분산형 DQN 모델에서는 같은 에피소드를 수행하였지만, 최대 큐함수 값이 6.83으로 다른 두 모델에 비해 매우 우수함을 볼 수 있었다. 이는 각각의 러너가 4,000회씩, 총 8,000회의 에피소드를 수행하였고, 이를 통하여 리플레이 메모리를 빠른 속도로 갱신하여 비교적 최신의 데이터로 학습을 수행하였기 때문이다.

그림 7.은 일정 시간 동안 학습한 에피소드 수에 따른 큐함수 평균값이다. 그림 7.의 (a)는 기존의 DQN 모델로 그림 7.의 (b)인 러너를 1개로 한 분산형 DQN 모델과 비슷한 학습률로 진행됨을 보여주었다. 그러나 기존의 DQN 모델이 5,000회의 에피소드를 수행할 동안에 러너를 1개로 한 분산형 DQN 모델은 5,000회가 넘는 에피소드 진행을 보여주어 학습과 행동을 분리한 분산형 DQN 모델이 학습 속도의 측면에서 개선됨을 알 수 있었다.

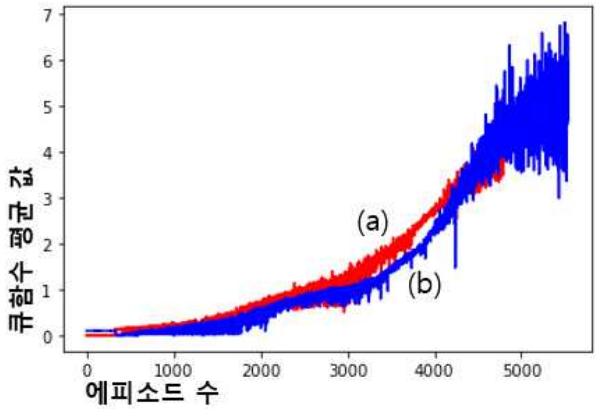

그림 7. (a) 기존의 DQN 모델, (b) 러너를 1개로 한 분산형 DQN 모델

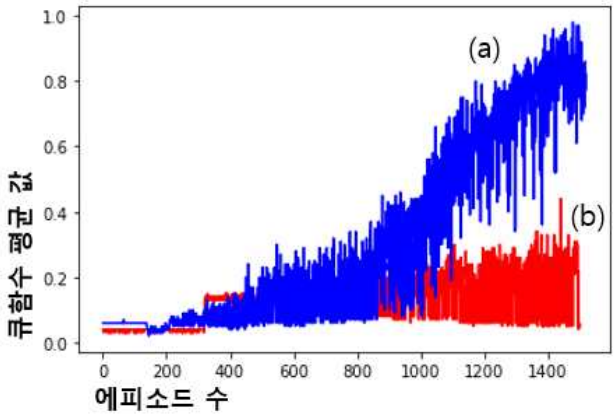

그림 8. (a) 러너를 2개로 한 분산형 DQN 모델, (b) 기존의 DQN 모델

그림 8.은 러너를 2개로 한 분산형 DQN 모델과 기존의 DQN 모델의 학습률의 초반 양상 그래프이다. 1,400회의 에피소드를 진행할 동안 그림 8.의 (a)인 러너를 2개로 한 분산형 DQN 모델은 약 800회의 에피소드부터 학습률이 급격하게 증가함을 보여주었지만, 그림 8.의 (b)인 기존의 DQN 모델은 1,400회의 에피소드 동안 학습률이 증가되지 않았음을 보여주어 제안한 DQN 모델이 학습률 측면에서 개선됨을 알 수 있었다.

이를 통하여 본 논문에서 제안한 학습과 행동 분산형 DQN 모델이 기존의 DQN 모델보다 학습률과 학습 속도에 있어서 개선됨을 확인하였다.

5. 결론

인공지능 분야에서 강화학습은 에이전트가 외부 환경과 상호작용하면서 목표를 달성하는 목표 지향적 기계학습 방법이다. 외부 환경이 매우 복잡하다면 행동에 대한 보상 예측값인 큐함수를 인공신경망으로 근사하는데, DQN은 리플레이 메모리와 타겟 모델로 이를 구현하였다. 그러나 기존의 DQN 모델은 리플레이 메모리를 사용한다는 점에서 저사양의 IoT 로컬 환경에 적용하기 어려우며, 과거의 데이터를 사용한다는 점은 학습 과정에서 방해 요소로 작용할 수 있다.

따라서 본 논문에서는 저사양의 로컬 환경에서 DQN을 사용할 수 있고, 과거의 데이터를 학습에 이용한다는 점을 개선하여 학습과 행동을 분리한 분산형 DQN 모델을 제안하였다. 모델은 글로벌 네트워크와 러너들로 구성되며 실제 BreakOut 게임 환경에서 기존의 DQN 모델과 비교한 결과 학습률과 학습 속도 측면에서 개선됨을 알 수 있었다. 그러므로 제안한 학습과 행동 분산형 DQN 모델은 고사양의 글로벌 네트워크와 저사양의 러너들로 구성될 수 있는 환경에서 기존의 DQN 모델에 비해 높은 학습률과 빠른 학습으로 고품질의 서비스를 제공할 수 있을 것으로 기대한다.

그러나 본 연구에서는 DQN 모델의 하이퍼 파라매터 튜닝을 실시하지 않았다. 향후 연구에서는 이를 고려하여 하이퍼 파라매터의 튜닝을 통해 분산 환경에서 좀 더 고품질의 서비스를 제공할 수 있도록 개선할 수 있을 것이다.

참고문헌

[1] Kim, Taeyoon, and Bonggyun Ko. "Modular reinforcement learning for dynamic portfolio optimization in the KOSPI market." The Korean Data & Information Science Society 32.1 (2021): 213-226.

[2] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533.

[3] Foerster, Jakob, et al. "Stabilising experience replay for deep multi-agent reinforcement learning." International conference on machine learning. PMLR, 2017.

[4] Szepesvári, Csaba. "Algorithms for reinforcement learning." Synthesis lectures on artificial intelligence and machine learning 4.1 (2010): 1-103.

[5] Li, Yuxi. "Deep reinforcement learning: An overview." arXiv preprint arXiv:1701.07274 (2017).

[6] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

[7] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[8] Bonawitz, Keith, et al. "Towards federated learning at scale: System design." arXiv preprint arXiv:1902.01046 (2019).

[9] Babaeizadeh, Mohammad, et al. "GA3C: GPU-based A3C for deep reinforcement learning." CoRR abs/1611.06256 (2016).

[10] https://github.com/KiHyeon-Hong/Improved-DQN-experiments