COSE-361 Final Project 2/2

박 준 영 2016130927

***Abstract*—파인 튜닝된 기존 베이스 라인 및 (Approximate) Q-Learning을 통한 학습**

***Index Terms*—Q-Learning, Approximate Q-Learning, Informed Search, Heuristic**

I. Introduction

주어진 베이스 라인은 각각 "더 많은 음식" 또는 "상대 팩맨의 존재 여부"만을 고려하고 있기 때문에 게임 내 다양한 상황을 고려하지 않는다.

현재 에이전트의 위치 정보에 따라 에이전트의 종류가 고스트/팩맨으로 달라지며, 이에 따라 고려할 특성 정보 역시 달라지기 때문이다. 따라서 세밀한 특성 별 가중치 값을 조절해 결과적으로 우선순위 큐에서 먼저 출력될 다음 핸동 역시 세밀하게 달라져야 한다. 베이스 라인을 정제하는 것만으로도 게임 상에서 유의미한 결과를 거둘 수 있었다.

파인 튜닝된 베이스 라인 알고리즘을 토대로 각각 Q-Learning, Approximate Q-Learning을 통해 상대방 영토에서 음식을 먹는 공격적인 에이전트의 행동을 교정하였다.

Q-Learning의 겨우 Approximate Q-Learning 알고리즘보다 정확한 행동 값을 도출할 수 있지만 도출하기까지 걸리는 비용(시간)이 다소 크기 때문에 결과적으로 Approximate Q-Learning이 전반적으로 더 뛰어난 결과를 보였다.

II. Method

A. baseline1: BaseAgent + DefensiveAgent

기존 베이스 라인 알고리즘의 특성 값과 가중치를 수정하는 것만으로도 유효한 결과를 거둘 수 있다.

현재 에이전트의 상황(팩맨/고스트)에 따라 다른 종류의 특성값을 주고, 특히 중요한 상황(상대방 팩맨이 침범했을 때/상대방 고스트로부터 위협받을 때)에 탈출할 수 있도록 특성값의 상수 값을 크게 높였다. (distanceToHome 등 특성값 가중치 참조)

(1). 현재 에이전트가 팩맨일 때: 1-1. 상대방의 영토에서 더 많은 음식을 먹어야 한다. 1.2. 상대방의 고스트로부터 달아나야 한다. 1.3. 캡슐을 통해 상대방의 고스트를 멈출 수 있다. 1.4. 자신의 영토로 돌아오는 길로 도망쳐야 한다. 1-5. 더 많은 점수를 올려야 한다.

(2). 현재 에이전트가 고스트일 때: 2-1. 상대방의 팩맨과 가까워야 한다. 2-2. 더 많은 점수를 올려야 한다.

위 특성 정보에 따른 가중치 값(남은 음식 개수, 가장 가까운 음식까지의 거리, 남은 캡슐 개수, 캡슐까지의 거리, 자신의 영토까지의 거리) 및 클래스의 특성 변수(쫓기고 있는지의 여부, 캡슐을 방금 먹었는지 여부 등)를 기존 베이스 라인 알고리즘에 추가했다.

파인 튜닝의 관건은 각 특성값에 따른 가중치 값을 어떻게 주느냐에 따라 달라지기 때문에, 가중치 값 수정만으로도 에이전트 행동 양상에 매우 다른 결과를 얻을 수 있다.

* 장점: 1. MDP 기반 모델에 비해 학습 속도가 빠르다. 2. 특성값과 가중치 조합에 따라 다양한 행동 반경이 제어 가능하다.

* 단점: 1. 적절한 가중치 휴리스틱을 구하는 게 성능의 가장 큰 관건이다. 2. Reflex 모델이기 때문에 모든 상황에 "똑똑하게" 대응하기 어렵다.

B. baseline2: AQLearningAgent + DefensiveReflexAgent

Q-Learning을 통해 현재 에이전트는 다음 단계에서 보다 유효한 행동을 학습할 수 있다. 보상 및 감마(디스카운트) 값을 통해 다음 Q 값을 어떻게 업데이트할지 조정할 수 있으며, 훈련 수(# of training) 등 기타 하이퍼 파라미터 역시 존재하지만, 본 코드에서는 고려하지 않았다.

강화학습을 통해 팩맨의 행동 알고리즘을 교정했다. 모델이 존재하지 않는다고 가정, 적절한 최적의 정책을 찾기 위한 온라인 학습을 시도했다.

Direct Evaluation, Temproal Difference Learing 등이 존재하지만 Q 학습을 통해 직접 최적의 정책을 학습하고자 했다.

QLearningAgent 클래스는 주어진 보상 체계를 통해 반복적으로 행동 원칙을 학습하며, 취할 수 있는 모든 행동을 self.q_value 딕셔너리에 기록한다.

현재 에이전트가 행동을 고를 때 주어진 값에 따라 Q 학습과 랜덤 행동을 선택(동전 플립)하게 된다. 다음 Q값을 최댓값으로 만들어주는 정책을 얻어낸 뒤, 하이퍼 파라미터 학습률에 따라 얻어낸 샘플을 어느 정도로 학습할지 결정한다. 수식으로 표현하면 다음과 같다.

$$Q^*(s,a) = \Sigma_{s'} T(s,a,s')[R(s,a,s') + {}^*(s')]$$

코드 상에는 self.update 함수를 통해 구현되어 있다. self.reward를 통해 다음 행동에 대한 보상을 추출, 이전에 기록한 self.q_value를 활용해 Q값의 최대로 이어지는 정책 a'을 찾고, 샘플을 얻어낼 수 있다.

$$sample \text{¸} R(s,a,s') + max_{a'} Q(s',a')$$

위 식을 정리하면 다음과 같다.

$$Q(s,a) \text{¸} (1-\alpha)Q(s,a) + \alpha * sample$$

탐색 시간이 충분하다면 학습률은 충분한 속도로 감소하고, 모든 Q 상태의 최적의 Q 값을 얻을 수 있다는 점에서 뛰어나다. 특히 온라인 학습의 다른 방법인 DE, TD 학습이 정책 결정 전에 정책을 결정해야 하는 반면, 직접적으로 최적의 정책을 학습할 수 있다.

* 장점: 시간이 충분하다면 Reflex 알고리즘보다 최적의 정책을 학습 가능하다.

* 단점: 1. 모든 행동에 대한 Q 값을 기록하기 때문에 학습 속도가 상당히 느리다. 2. 하이퍼 파라미터(학습률,

디스카운트), 보상 함수 등 사전 설계에 따라 학습 효과가 달라진다. 3. 메모리 낭비가 심하다.

실제로 QLearingAgent로 유효한 결과를 거두는 데에는 실패했다. 기존 BaseAgent에 비해 학습 속도가 매우 느렸기 때문이다. QLearningAgent를 보완한 Approximate Q-Learning을 통해 빠른 속도로 정책을 학습할 수 있었고, 효과를 거두었다.

$$difference = [R(s,a,s') + \gamma max_{a'} Q(s',a')] - Q(s,a)$$

기존의 difference를 사용, 특성값에 곱한 값을 통해 가중치를 업데이트한다. 이때 사용되는 특성값은 이전의 self.q_value를 사용해 모든 행동의 상태 값을 기록해둔 바와 달리 일반적인 경우를 상정하고 있다.

$$w_i \leftarrow w_i + \alpha \cdot difference \cdot f_i(s,a)$$

이렇게 빠른 속도로 업데이트한 가중치 정보를 통해 이후 Q 값을 효과적으로 (빠르게) 업데이트할 수 있다.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot difference$$

* 장점: Q 학습에 비해 정책 학습이 매우 빠르다.

* 단점: Q 학습과 마찬가지로 문제 상황이 바뀐다면 주어진 보상 체계에 따라 학습 효과가 달라지게 된다.

AQLearingAgent를 통해 Offensive 모델을 구현했고, Defensive 모델로 기존의 베이스 라인 DefensiveReflexAgent를 택했다.

C. baseline3: AQLearningAgent + TimidAgent

baseline2와 마찬가지로 AQLearningAgent를 통해 Offensive 모델을, BaseAgent를 수정, 방어에 더 많은 가중치를 둔 TimidAgent 모델을 사용했다. TimidAgent는 기존 BaseAgent의 현재 상황이 팩맨일 때 상대방 고스트로부터의 최단 거리가 5일 때 가중치를 두어 돌아가는 것과 반대로, 가중치가 10일 때 되돌아가는 다소 "신중한" 모델이다.

* 장점: 기존 베이스라인 에이전트에 대해 매우 높은 확률로 승리를 거둔다.

* 단점: baseline1(BaseAgent + DefensiveAgent)에게 매우 큰 확률로 패배한다. TimidAgent는 상대방과 거리가 가까워질 때 매우 빠른 속도로 멀어지기 때문에 다소 공격적인 baseline1에 비해 점수를 얻을 확률이 낮다. 아래 결과를 확인할 것.

your_best(red)	
<Average Winning Rate>	
your_base1	-0.7
your_base2	0.6
your_base3	0.7
baseline	1
Num_Win	3
Avg_Winning_Rate	0.4
<Average Scores>	
your_base1	-0.6
your_base2	0.4
your_base3	0.2
baesline	3.4
Avg_Score	0.85

D. best alogirithm: AQLearningAgent + DefensiveAgent

결과적으로 구현한 에이전트 모델 중 Offensive로는 AQLearningAgent를, Defensive로는 BaseAgent를 개량한 DefensiveAgent를 택했다.

* 장점: 1. 전반적으로 baseline3을 제외한 나머지 세 종류의 에이전트 모델에게 높은 확률로 승리를 거둔다. 2. DefensiveAgent는 상대방의 팩맨을 고려하는 가중치뿐만 아니라 음식까지 고려하는 양방향 모델이기 때문에 점수를 더 높게 얻을 확률이 높다. * 단점: 1. TimidAgent를 Deffensive 모델로 택한 baseline3에게 다소 높은 확률로 패배한다. 2. 장점2의 역기능으로, 상대방의 두 에이전트 모두가 공격적이라면 점수를 크게 빼앗길 확률이 높을 것이다.

III. Result

	your_best(red)
	<Average Winning Rate>
your_base1	0.6
your_base2	0.7
your_base3	-0.6
baseline	0.9
Num_Win	3
Avg_Winning_Rate	0.4
	<Average Scores>
your_base1	-0.7
your_base2	2.5
your_base3	0
baesline	2.7
Avg_Score	1.125

IV. Conclusion

에이전트 모델을 구현하며 보다 개선할 점은 다음과 같다.

1. 상대방 에이전트가 먹은 캡슐에 따라 상황이 반전될 때를 고려해야 한다. 2. 게임 시간이 끝날 때 더 많은 음식을 먹기보다는 자신의 영토로 돌아가야 한다. 3. Offensive 모델이 ”더 가까운” 음식으 먹기보다 현 시점에서 ”접근성이 가장 높은” 음식을 찾는 데 가중치를 줄 수 있다. 4. 현재 에이전트가 팩맨이고 추격당하고 있는 위험한 상황일 때 캡슐을 사용하는 전략이 유효할 것이다. 5. Defensive 모델이 상대 에이전트를 방어하는 시점은 상대 팩맨이 ”나갈 때”로 충분할 것이다. 6. 많은 특성보다 주요 특성에 가중치를 더 주는 게 효과적일 수 있다. 연산 시간 역시 에이전트의 반응 속도와 얼마나 연관되는지 확인해야 한다.

결과적으로, 강화학습을 통해 현재 에이전트가 최적의 정책을 찾아가는 방법을 깊이 있게 다룰 수 있었다. 이때 학습률, 디스카운트 등 하이퍼 파라미터뿐만 아니라 특성을 제어하고 보상을 어느 정도로 줄지 사전 정의가 매우 큰 영향을 미칠 수 있음 또한 알 수 있었다.