인공지능 오목

강화 학습에 대한 아주 기초적인 이야기는 <https://greentec.github.io/reinforcement-learning-first/>

DQN에 관한 이야기는 <https://greentec.github.io/reinforcement-learning-second/>

다양한 DQN method에 관한 이야기는 <https://greentec.github.io/reinforcement-learning-third/>

Actor Critic에 관한 이야기는 <https://greentec.github.io/reinforcement-learning-fourth/>

오목 프로그램은 Actor Critic 방식을 차용해서 만들어졌습니다.

오목 프로그램의 기초 Monte Carlo tree search. (<https://en.wikipedia.org/wiki/Monte_Carlo_tree_search>)

몬테카를로 트리 탐색은 4가지 단계로 이루어집니다. Select expand, simulate back propagation.

Select 단계에서는 어떠한 node를 선택하는 지를 설정합니다. 이때 설정 방식은, 방문한 횟수와 보상의 상관관계를 사용합니다. 처음에는 방문한 횟수가 적은 것을 선택하고, 나중에 되면, 방문한 횟수가 높은 것을 선택하거나, 보상의 평균이 높은 것을 선택하는 방식으로 이루어져 있습니다. 이때 선택된 노드는 leaf node입니다.

Expand는 선택한 노드를 확장하는 것을 의미합니다. Leaf node를 확장한다음, 그 중 가지고 있는 MCTS의 policy로 하나의 값을 선택합니다.

Simulation은 선택한 노드를 다양한 rollout policy를 사용해서 실험합니다. 이때 실험을 통해서 얻은 reward를 저장합니다.

Back propagation은 simulation을 통해서 얻은 reward를 tree의 root까지 전달하는 것으로, 나중에 이를 통해서 전체적인 노드의 추세를 확인할 수 있습니다.

(main에서 playTicTacToe 함수를 통해서 MCTS의 가장 기본적인 구조를 확인할 수 있습니다.)

Alpha Zero는 MCTS와 DQN을 기반으로 만들어졌습니다. MCTS의 rollout policy를 DQN에서 얻는 Value 값으로 치환하는 방식입니다.

DQN을 통해서 구한 Q value 값과 각 node를 방문한 횟수를 사용하여 하나의 함수를 구합니다. 이 함수를 통해서 구한 값을 사용하여 어떠한 노드를 더 방문할지를 선택합니다. 이때 방문 방식은 value + policy/number of root visit으로 구성되어 있습니다. 처음에는 policy에 따라서 방문을 하지만, 나중으로 갈수록, reward가 높아지는 node를 선택하게 됩니다.

현제 오목 self play는 20초 정도 생각할 시간을 줍니다. 20초를 사용해서 여러가지 방면으로 expand를 하고 이때 방문한 Node의 값을 probability로 변환해 저장합니다. 이 저장된 값은 나중에 train할 때 사용합니다. 또한 게임이 끝난 reward 부터 감가율을 적용하여 저장합니다. 이 또한 나중에 DQN을 train하는데 사용합니다.

처음 train을 할 때는 50개의 node를 방문했고, 그 다음은 64개의 node를 방문했으며, 지금은 800회 방문을 하고 있습니다. 처음에 적은 수의 node를 사용한 이유는 게임의 양을 늘리기 위함입니다. 800회 방문을 하는 이유는 게임의 질을 높이기 위함입니다.

현제 노드 선택 방식은, 방문한 횟수를 전체 횟수로 나는 확률에 따라서 선택하게 됩니다. 예를 들어 총 100회 방문했다고 가정할 때, 하나의 노드를 4 번 방문했다고 가정하면, 이 노드를 선택할 확률은 4/100, 4%의 확률로 선택하게 됩니다.

나중 게임을 직접 play 할 때에는 Deterministic policy를 사용합니다. Deterministic policy는 위에서 확률적으로 선택한 값을 사용하는 것이 아닌, 가장 많은 노드를 선택하는 것으로 대체됩니다.

Training은 self play를 통해서 얻은 probability와 reward 값을 사용해서 구하게 됩니다. 현제 트레이닝 방식은 2048개의 batch size를 통해서 훈련을 하고 있습니다. 그리고 한 epoch 당 1000개의 step으로 train하고 있습니다. 한 training 당 걸리는 시간은 대략 40분정도 이며, 대략 16\*30개의 개임이 지나야 업데이트 되는 것 같습니다.

이제 코드의 변경은 없이 self play와 training을 무한히 반복하면, DQN이 좋은 policy와 value를 predict하는 것을 예상해봅니다. 하지만, KataGo 논문에서 이야기한 것처럼, self play를 통해서 좋은 성적을 얻으려면, 엄청나게 많은 시간이 필요할 것으로 예상이 됩니다.

Reference

1. David silva Reinforcement Learning by DeepMind

2. https://arxiv.org/pdf/1712.01815.pdf (알파 제로 논문)

3. https://www.nature.com/articles/nature24270.epdf?author\_access\_token=VJXbVjaSHxFoctQQ4p2k4tRgN0jAjWel9jnR3ZoTv0PVW4gB86EEpGqTRDtpIz-2rmo8-KG06gqVobU5NSCFeHILHcVFUeMsbvwS-lxjqQGg98faovwjxeTUgZAUMnRQ (알파고제로 논문)

4. https://dl.fbaipublicfiles.com/elfopengo/pdf/arxiv.pdf (ELF go 논문)

5. https://storage.googleapis.com/deepmind-media/alphago/AlphaGoNaturePaper.pdf (알파고 논문)

6. https://arxiv.org/pdf/1902.10565.pdf (KataGo 논문)

7. https://arxiv.org/pdf/1809.10595.pdf (알파 오목 논문)

8. https://sumniya.tistory.com/18?category=781573 (DQN 설명)

9. https://greentec.github.io/reinforcement-learning-first/ (강화학습 설명)

10. https://web.stanford.edu/~surag/posts/alphazero.html (알파제로 논문 설명)