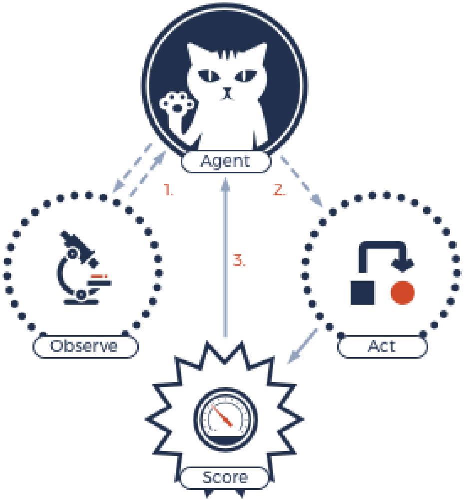
Deep Reinforcement Learning

Carrot or stick?

Reinforcement Learning



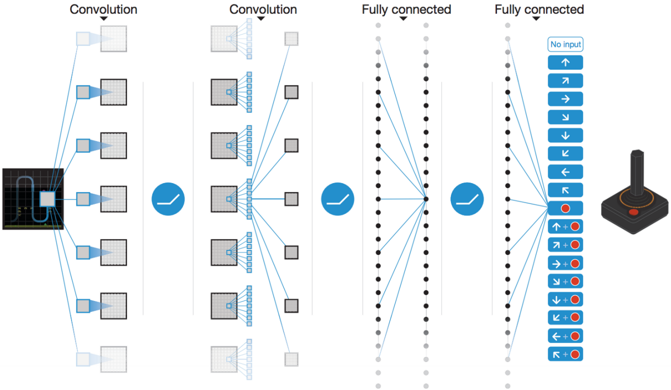
Bellman Equations

$$
\begin{aligned}
V^{\pi}(s_t) &= \mathrm{E}\left[r(s_t, a_t) + \gamma V^{\pi}\left(s_{t+1}\right)\right], \\
Q^{\pi}(s_t, a_t) &= \mathrm{E}\left[r(s_t, a_t) + \gamma \mathrm{E}\left[Q^{\pi}\left(s_{t+1}, a_{t+1}\right)\right]\right].
\end{aligned}
$$

Types of RL

- Model-free methods
  - Policy optimization methods
  - Q learning methods
- Model-based methods
  - Learned model
  - Given model

Deep Q Network



Loss Function

$$
\begin{aligned}
L(s,a) &= \mathbb{E}\left[\left(\hat{Q}(s,a) - Q(s,a)\right)^2\right], \\
&= \mathbb{E}\left[\left(r(s_t,a_t) + \gamma \max_{a_{t+1}} Q\left(s_{t+1},a_{t+1}\right) - Q(s,a)\right)^2\right].
\end{aligned}
$$

Training Considerations

- Experience Replay
- Fixed Q Targets
- Reward Clipping
- Skipping Frames

Experience Replay

- The network stores or caches experiences and the resulting values without applying any learning to them then it learns weights for its experiences all at once.
- Helps with overfitting

Fixed Q Targets

- Weights are only updated once every f iterations
- Helps stabilize the model

Reward Clipping

- Clip the rewards to be either 1 or -1.
- Helps with standardizing across environments

Skipping Frames

- Skips k frames in state and whatever previous action was determined is applied to those frames.
- Speeds up training and playing

Demo

Problems with Q Learning

- Large state and action space
- Difficulty generalizing

Policy Optimization

Policy Definition

$$\pi_\theta(s, a) = P(a|s, \theta),$$

Policy Value

$$J(\theta) = \mathsf{E}\left[\sum_{t=0}^{H} R\left(s_t, u_t\right); \pi_\theta\right] = \sum_{\tau} P(\tau;\theta)R(\tau),$$

where

$$R(\tau) = \sum_{t=0}^{H} R\left(s_t, u_t\right).$$

.

Polcy Gradient

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) \\
&= \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_\tau P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) \\
&= \sum_\tau P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)
\end{aligned}
$$

Source

Vanilla Policy Gradient Optimization

General idea is to

1. Initialize parameters
2. Run a few test runs with the policy and collect data about the rewards
3. Estimate the policy gradient
4. Update policy using gradient ascent
5. Compute loss function and repeat from step 2

Actor-Critic Approach

Combine policy optimization approaches with Q learning approaches.

Actor-Critic

Actor = policy optimization method, Critic = Q learning method

- Actor selects actions
- Critic returns values
- Actor updates its action selection policy

Actor-Critic Benefits

- Combines benefits of both policy optimization and Q learning
- Q learning method only tracks actions that are chosen by the policy

Walking Demo

Traffic Demo

Steering Demo

Questions