Deep Reinforcement Learning Lecture

Hado van Hasselt

DeepMind

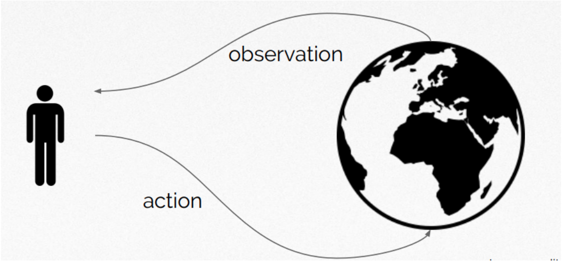
Presenter: Jack Lanchantin

1. Introduction
2. Deep Q Learning
   - Q-learning
   - Deep Q Learning
3. Policy Gradients

Outline

Reinforcement Learning

- RL provides a general-purpose framework for making decisions
  - RL is about learning to act
  - Each action can alter the state of the world, and can result in reward
  - Goal: optimize future rewards (which may be internal to the agent)



- Used on problems that involve making decisions and/or making predictions about the future

Approaches to reinforcement learning

- The goal is to learn a policy of behaviour
- (At least) three possibilities:
  - Learn policy directly
  - Learn values of each action - infer policy by inspection
  - Learn a model - infer policy by planning
- Agents therefore typically have at least one of these components:
  - Policy - maps current state to action
  - Value function - prediction of value for each state and action
  - Model - agents representation of the environment.

Policy

- A policy is the agent's behaviour
- It is a map from state to action:
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \mathbb{P}[a|s]$

Value Function

- A value function is a prediction of future reward
  - "How much reward will I get from action a in state s?"
- Q-value function gives expected total reward
  - from state s and action a
  - under policy π
  - with discount factor γ

$$Q^{\pi}(s, a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right]$$

- Value functions decompose into a Bellman equation

$$Q^{\pi}(s, a) = \mathbb{E}_{s', a'}\left[r + \gamma Q^{\pi}(s', a') \mid s, a\right]$$

Optimal Value Function

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) = Q^{\pi^*}(s,a)$$

- Once we have Q^* we can act optimally,

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

- Optimal value maximises over all decisions. Informally:

$$\begin{aligned} Q^*(s,a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

- Formally, optimal values decompose into a Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a \right]$$

Reinforcement learning Components

- Policy: $\pi(s) = a$
- Value: $Q(s, a) \approx \mathbb{E}[R_{t+1} + R_{t+1} + ...|S_t = s, A_t = a]$
- Model: $m(s, a) \approx \mathbb{E}[S_{t+1}|S_t = s, A_t = a]$

$\rightarrow$ We need to represent and learn these functions

Approaches to RL

Value-based RL

- Estimate the optimal value function $Q^*(s, a)$
- This is the maximum value achievable under any policy

Policy-based RL

- Search directly for the optimal policy π^*
- This is the policy achieving maximum future reward

Model-based RL

- Build a model of the environment
- Plan (e.g. by lookahead) using model

Deep reinforcement learning

Use deep learning to learn policies, values, and/or models to use in a reinforcement learning domain

- Reinforcement learning provides: a framework for making decisions
- Deep learning provides: tools to learn the components

Outline

Q-learning: A algorithm to learn values

- The optimal value function fulfills:

$$Q^*(s,a) = \mathbb{E}[R_{t+1} + max_{a'} Q^*(s', a') | \, s, a \,] \tag{1}$$

  i.e. the value of the policy that will get you the most reward

- We can turn this into a temporal difference algorithm

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha \big(R_{t+1} + \gamma max_a Q_t(S_{t+1}, a) - Q_t(S_t, A_t) \big) \tag{2}$$

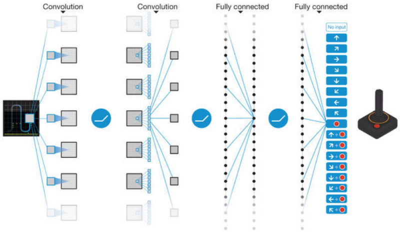
Q-learning

- By learning off-policy about the policy that is currently greedy, Q-learning can approximate the optimal value function Q^*
- With Q^* we have an optimal policy: $\pi^*(s) = argmaxQ^*(s, .)$

Deep Q Network (Mnih et al., Nature 2015)

- Learns to play video games by simply playing and observing rewards
- Can learn the Q function by Q-learning

$$\Delta w = \alpha\big(R_{t+1} + \gamma max_a Q(S_{t+1}, a; w) - Q(S_t, A_t; w)\big)\nabla_w Q(S_t, A_t; w)$$



Target Networks

- Changing the value of one action will change the value of other actions and similar states
- The network can end up chasing its own tail because of bootstrapping
- Solution: freeze the weights in the target network for K number of update steps

$$\Delta w = \alpha \big(R_{t+1} + \gamma max_a Q(S_{t+1}, a; w^{-}) - Q(S_t, A_t; w) \big) \nabla_w Q(S_t, A_t; w)$$

Experience Replay

- Replay previous tuples (s,a,r,s') which the agent has seen before
- Benefits:
  - More data efficient
  - Learning resembles supervised learning more (which deep learning works well on)
- Replay can be sampled in specific ways, e.g. replay transitions in proportion to absolute Bellman error:

$$|r + \gamma max_{a'} Q(S', a', w) - Q(s, a, w)| \tag{3}$$

Double DQN (van Hasselt et al. 2015)

DQN:

$$\begin{aligned}\Delta w &= \alpha\Big(r_{t+1} + \gamma max_{a'} Q(s', a'; w^{-}) - Q_t(s, a; w)\Big)\nabla_w Q(s, a; w)\\ &= \alpha\Big(r_{t+1} + \gamma Q(s', argmax_{a'} Q(s', a'; w^{-}); w^{-}) - Q_t(s, a; w)\Big)\nabla_w Q(s, a;\end{aligned}$$

Double DQN:

$$\Delta w = \alpha\Big(r_{t+1} + \gamma Q(s', argmax_{a'} Q(s', a'; w); w^{-}) - Q_t(s, a; w)\Big)\nabla_w Q(s, a; w)$$

Main Idea: decorrelate selection and evaluation to mitigate overestimation

Outline

Policy Gradient

- We can often do better if the policy is differentiable (optimize the performance with SGD).
  - Represent policy by deep network with weights θ: $a = \pi(a|s, \theta)$
  - Adjust policy parameters θ to achieve more reward
- Goal: compute gradient of the following objective:

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + ...|\pi(\cdot, \theta)] \tag{4}$$

- Problem: rewards aren't differentiable $\rightarrow$ estimate the gradient

Policy Gradient Theorem

- For all differentiable policies (where expectation is over all states and actions):

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta log \pi_{\theta_t}(a|s) Q^\pi(s,a)] \tag{5}$$

there is an easy sample-based approximation (REINFORCE):

$$\nabla_\theta log \pi_{\theta_t}(a_t|s_t) G_t$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$$

Update parameters:

$$\theta_{t+1} = \theta_t + \alpha R_{t+1} \nabla_\theta log \pi_{\theta_t}(a_t|s_t) G_t \tag{6}$$

Practical Deep Policy Gradient

- How can policy-based methods be implemented efficiently with neural networks?
- DQN uses replay, but standard PG methods are on-policy
    - Good off-policy PG methods have since been developed: ACER (Wang et al., 2016) and PGQL (ODonoghue et al., 2016)
    - Idea: sample from replay, but adapt the updates so that expected gradient looks as if we use the current policy

Conclusion

- RL: general framework for learning how to act in an environment
- DL: tool to learn the policy of how to act (either through value or policy iteration)