Mobile Robotics, Reinforcement Learning and Deep Reinforcement Learning for Mobile Robots

Material based on Reinforcement Learning: an Introduction, 2nd Edition [sect. 5.1-5.3, 6.1-6.3, 6.5], Reinforcement Learning course offered by Prof. Pascal Poupart at Univ. of Waterloo

Summary

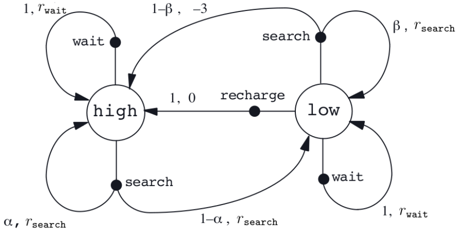
- Introduction to Reinforcement Learning
- Deep Reinforcement Learning
- Deep Q Network
- DQN for Mapless Navigation

Reinforcement Learning: relationships with MDPs

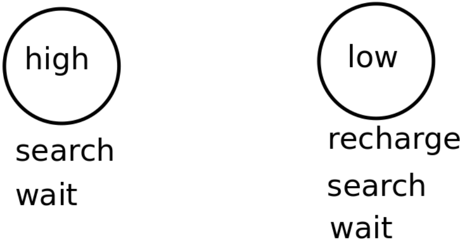
Guide an MDP without knowing the dynamics

- do not know which states are good/bad (no $R(s,a)$)
- do not know where actions will lead us (no $T(s,a,s')$)
- hence we must try out actions/states and collect the reward

Recycling robot example: RL



Planning



Learning

To use a model or not to use a model ?

- Model-Based methods try to **learn a model**
  - + avoid repeating bad states/actions
  - + fewer execution steps
  - + efficient use of data
- Model-Free methods try to **learn Q-function and policy** directly
  - + simplicity, no need to build and use a model
  - + no bias in model design

Q-Learning: pseudo-code

Algorithm 1 Tabular Q-Learning

```
Initialize Q(s, a) arbitrarily
Initialize s {observe current state}
loop
    Select and execute action a
    Observe new state s' receive immediate reward r
    Q(s, a) = Q(s, a) + α[r + γ max'_a Q(s', a') − Q(s, a)]
    update state s ← s'
end loop
```

◇ ϵ-greedy: choose best action most of the time, but every once in a while (with probability ϵ) choose randomly amongst all action (with equal probabiliy)

Deep Reinforcement Learning: key points

- ◇ For many real world domains we can not explicitly represent key functions for RL ($\pi(s)$, $V(s)$, $Q(s,a)$)
- ◇ We can try to approximate them
  - Linear approximation
  - Neural Network approximation
    - Deep RL
- ◇ Deep Q Network approximates $Q(s,a)$ with a DNN

Gradient Q-Learning

- approximate $Q(s, a)$ with a parametrized function $Q_{\mathbf{w}}(s, a)$
- Minimize squared error between estimate and target
  - Estimate $Q_{\mathbf{w}}(s, a)$
  - Target: $r(s, a, s') + \gamma \max_{a'} Q_{\overline{\mathbf{w}}}(s', a')$
- squared error:

$$Err(\mathbf{w}) = (Q_{\mathbf{w}}(s, a) - r(s, a, s') - \gamma \max_{a'} Q_{\overline{\mathbf{w}}}(s', a'))^2$$

- gradient:

$$\frac{\partial Err(\mathbf{w})}{\partial \mathbf{w}} = 2(Q_{\mathbf{w}}(s, a) - r(s, a, s') - \gamma \max_{a'} Q_{\overline{\mathbf{w}}}(s', a'))\frac{\partial Q_{\mathbf{w}}(s,a)}{\partial \mathbf{w}}$$

(Scalar 2 is a constant factor and not important for update)

Gradient Q-Learning Algorithm

Algorithm 2 Gradient Q-Learning

```
Initialize weights w randomly in [−1, 1]
Initialize s {observe current state}
loop
    Select and execute action a
    Observe new state s′ receive immediate reward r
    ∂Err(w)/∂w = (Q_w(s, a) − r − γ max_a′ Q_w(s′, a′)) ∂Q_w(s,a)/∂w
    update weights w ← w − α ∂Err(w)/∂w
    update state s ← s′
end loop
```

$$\frac{\partial Err(\mathbf{w})}{\partial \mathbf{w}} = (Q_{\mathbf{w}}(s,a) - r - \gamma \max_{a'} Q_{\mathbf{w}}(s',a'))\frac{\partial Q_{\mathbf{w}}(s,a)}{\partial \mathbf{w}}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial Err(\mathbf{w})}{\partial \mathbf{w}}$$

Non-Convergence of Non-linear gradient Q-Learning

- Non-linear approximation of Q(s,a), $Q(s,a) \approx g(\mathbf{x}; \boldsymbol{w})$
- **gradient Q-Learning may not converge**
- Issue:
  - we update the weights to reduce error for a specific experience (i.e., a specific (s,a)) **but** by changing the weights we may end up changing the $Q(s,a)$ potentially everywhere.

Mitigating divergence

- Two main approaches to **mitigate** divergence:
  1. experience replay
  2. use two different networks
     - Q-network
     - Target network

Experience replay

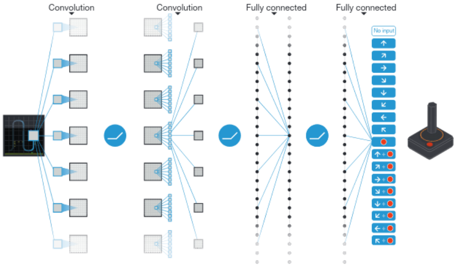
- Store previous experiences (i.e., (s, a, s', r)) and use them at each step
  - Store previous (s, a, s', r) in a dedicated memory buffer
  - At each step sample a mini-batch from this buffer and use the mini-batch to update the weights
- Benefits
  1. reduces correlation between successive samples (increase stability)
  2. reduces number of interaction with the environment (increase data efficiency)

Target Network

- Maintain a separate **target** network and update this network periodically (not with every experience)
  - Q-network $Q_{\boldsymbol{w}}(s, a)$
  - Target network $Q_{\overline{\boldsymbol{w}}}(s, a)$
- repeat for every (s, a, s', r) in the mini-batch update the Q-network
  - $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_t(Q_{\boldsymbol{w}}(s, a) - r - \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s', a'))\frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$
- update the target network
  - $\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

Deep Q Network

- Human-level control through deep reinforcement learning (V. Mnih et al., Nature 2015)
- Gradient Q-Learning
  - Deep neural networks to approximate Q(s,a)
  - Experience Replay and Target network



- above human-level performance in many Atari video games

Deep Q Network sketch of algorithm

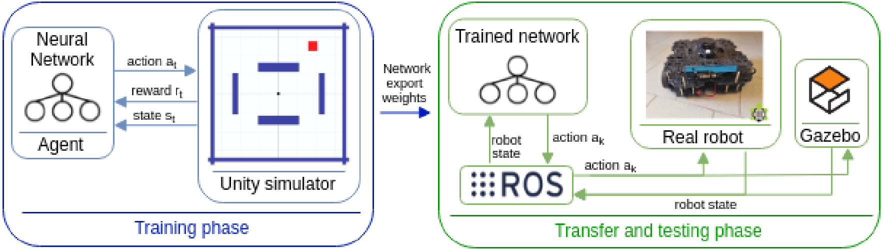
Algorithm 3 DQN

Initialize weights $\boldsymbol{w}$ and $\overline{\boldsymbol{w}}$ randomly in $[-1, 1]$
Initialize s {observe current state}
loop
    Select and execute action a
    Observe new state s' receive immediate reward r
    Add (s, a, s', r) to experience buffer
    Sample mini-batch MB of experiences from buffer
    for $(\hat{s}, \hat{a}, \hat{s}', \hat{r}) \in MB$ **do**
        $\frac{\partial Err(\boldsymbol{w})}{\partial \boldsymbol{w}} = (Q_{\boldsymbol{w}}(\hat{s}, \hat{a}) - \hat{r} - \gamma \max_{\hat{a}'} Q_{\overline{\boldsymbol{w}}}(\hat{s}', \hat{a}')) \frac{\partial Q_{\boldsymbol{w}}(\hat{s}, \hat{a})}{\partial \boldsymbol{w}}$
        update weights $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial Err(\boldsymbol{w})}{\partial \boldsymbol{w}}$
    end for
    update state $s \leftarrow s'$
    every c steps, update target: $\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$
end loop

DRL for robotics, sim to real

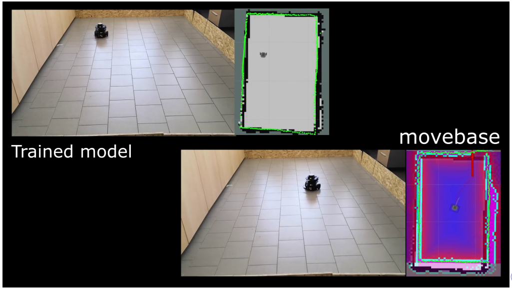
Acting in the real environment is difficult/dangerous

- train in a synthetic environment



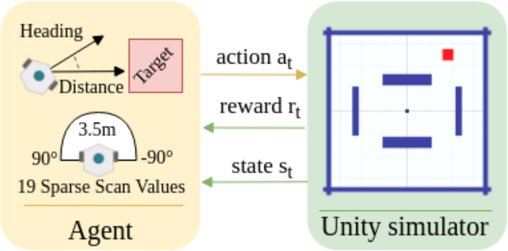
DRL for robotics, continuous approaches and mapless navigation

- ♢ Most DRL approaches considers continuous action space
  - Proximal Policy Optimization (PPO)
  - Policy Gradient (DDPG)
- ♢ Continuous approaches are time-consuming compared to discrete action space (e.g., DQN)
- ♢ **Mapless Navigation**: Navigate in the environment, avoiding obstacles, without a map
  - laser scans and target heading as input
  - angular velocities as output
  - dense reward (i.e., distance from target)



Discrete DRL for mapless navigation

Proper discretization of the action space and the use of DQN can result in a significantly shorter training time (from 20 hours to 1 hour), maintaining comparable performance.



Deep RL: current trends

- Formal verification of DRL models
  - ensure the learned model respect safety properties
- Transfer of Learning/Curricula learning
  - ToL: train the model in an environment and deploy in another one
  - Curricula Learning: learn a difficult task by training on a series of simpler tasks
- DRL for robotics
  - Adaptation to environment is critical but interacting with the environment is difficult, expensive and potentially dangerous.
- MADRL
  - A set of agents/robots that learn at the same time in the same environment

- **PPO**: Schulman et al. (2017) Proximal Policy Optimization
- **DDPG**: Lillicrap et al. (2015) Continuous control with deep reinforcement learning
- **Mapless**: Tai et al. (2017) "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation"