Introduction to deep reinforcement learning
Outline

• Introduction to reinforcement learning
• Markov Decision Process (MDP) formalism
• The Bellman equation
• Q-learning
• Deep Q networks (DQN)
• Extensions
  • Double DQN
  • Dueling DQN
Reinforcement learning (RL)

• Setting: agent that can take *actions* affecting the *state* of the environment and observe occasional *rewards* that depend on the state
• Goal: learn a *policy* (mapping from states to actions) to maximize expected reward over time
RL vs. supervised learning

- **Reinforcement learning loop**
  - From state $s$, take action $a$ determined by policy $\pi(s)$
  - Environment selects next state $s'$ based on transition model $P(s'|s,a)$
  - Observe $s'$ and reward $r(s')$, update policy

- **Supervised learning loop**
  - Get input $x_i$ sampled i.i.d. from data distribution
  - Use model with parameters $w$ to predict output $y$
  - Observe target output $y_i$ and loss $l(w, x_i, y_i)$
  - Update $w$ to reduce loss: $w \leftarrow w - \eta \nabla l(w, x_i, y_i)$
RL vs. supervised learning

- **Reinforcement learning**
  - Agent’s actions affect the environment and help to determine next observation
  - Rewards may be sparse
  - Rewards are *not differentiable* w.r.t. model parameters

- **Supervised learning**
  - Next input does not depend on previous inputs or agent’s predictions
  - There is a supervision signal at every step
  - Loss is differentiable w.r.t. model parameters
Example applications of deep RL

• AlphaGo and AlphaZero

https://deepmind.com/research/alphago/
Example applications of deep RL

- Playing video games

Example applications of deep RL

- Sensorimotor learning

Fig. 1: Our method learns visuomotor policies that directly use camera image observations (left) to set motor torques on a PR2 robot (right).

Video

Example applications of deep RL

• Sensorimotor learning

Example applications of deep RL

- Improving large language models

L. Ouyang et al. *Training language models to follow instructions with human feedback*. NeurIPS 2022
Formalism: Markov Decision Processes

- Components:
  - **States** $s$, beginning with initial state $s_0$
  - **Actions** $a$
  - **Transition model** $P(s' | s, a)$
    - *Markov assumption*: the probability of going to $s'$ from $s$ depends only on $s$ and $a$ and not on any other past actions or states
  - **Reward function** $r(s)$
  - **Policy** $\pi(s)$: the action that an agent takes in any given state
    - The “solution” to an MDP
Example MDP: Grid world

\[ r(s) = -0.04 \] for every non-terminal state

Transition model:

Source: P. Abbeel and D. Klein
Example MDP: Grid world

• Goal: find the best policy

Source: P. Abbeel and D. Klein
Example MDP: Grid world

- Optimal policies for various values of $r(s)$:
Cumulative rewards of state sequences

- Suppose that following policy $\pi$ starting in state $s_0$ leads to a sequence or trajectory $\tau = (s_0, s_1, s_2, \ldots)$
- How do we define the cumulative reward of this trajectory?
- **Problem:** the sum of rewards of individual states grows is not normalized w.r.t. sequence length and can even be infinite
- **Solution:** define cumulative reward as sum of rewards *discounted* by a factor $\gamma$, $0 < \gamma \leq 1$

Image source: P. Abbeel and D. Klein
Discounting

• **Discounted cumulative reward** of trajectory $\tau = (s_0, s_1, s_2, s_3, ...)$:

\[
    r(s_0, s_1, s_2, s_3, ... ) = r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \gamma^3 r(s_3) + ... = \sum_{t \geq 0} \gamma^t r(s_t)
\]

• Sum is bounded by $\frac{r_{\text{max}}}{1 - \gamma}$ (assuming $0 < \gamma \leq 1$)

• Helps algorithms converge

• Notice:

\[
    r(s_0, s_1, s_2, s_3, ... ) = r(s_0) + \gamma r(s_1, s_2, s_3, ...)
\]

Cumulative reward of trajectory starting at $s_0$  
Reward at $s_0$  
Discounted reward of trajectory starting at $s_1$
Value function

• The value function $V^\pi(s)$ of a state $s$ w.r.t. policy $\pi$ is the expected cumulative reward of following that policy starting in $s$:

$$V^\pi(s) = \mathbb{E}_{\tau}[r(\tau) | s_0 = s, \pi]$$

where $\tau$ is a trajectory with starting state $s$, actions given by $\pi$, and successor states drawn according to transition model:

$s_{t+1} \sim P(\cdot | s_t, a_t)$

• The optimal value of a state is the value achievable by following the best possible policy:

$$V^*(s) = \max_\pi V^\pi(s)$$
The optimal policy

• How do we express the optimal policy in terms of optimal state values?
The optimal policy

- How do we express the optimal policy in terms of optimal state values?
The optimal policy

- How do we express the optimal policy in terms of optimal state values?
  - Take the action that maximizes the expected future cumulative value:
    \[ \pi^*(s) = \arg\max_a \mathbb{E}_{s' \sim P(\cdot | s, a)} V^*(s') \]
The Bellman equation

- Recursive relationship between optimal values of successive states:

\[ V^*(s) = r(s) + \gamma \max_a \mathbb{E}_{s' \sim P(\cdot|s,a)} V^*(s') \]
The Bellman equation

- Recursive relationship between optimal values of successive states:

\[ V^*(s) = r(s) + \gamma \max_a \mathbb{E}_{s' \sim P(\cdot|s,a)} V^*(s') \]

- Reward in current state
- Discounted expected future reward assuming agent follows the optimal policy
Outline

• Introduction to reinforcement learning
• Markov Decision Process (MDP) formalism
• The Bellman equation
• Q-learning
Q-learning

• To choose actions using value functions, we need to know the transition model:

\[ \pi^*(s) = \arg \max_a \mathbb{E}_{s' \sim P(s'|s,a)} V^*(s') \]
\[ = \arg \max_a \sum_{s'} P(s'|s,a) V^*(s') \]

• It is more convenient to define the value of a state-action pair:

\[ Q^\pi(s,a) = \mathbb{E}_\tau [r(\tau)|s_0 = s, a_0 = a, \pi] \]

• Then the optimal policy is given by

\[ \pi^*(s) = \arg \max_a Q^*(s,a) \]
Q-value function: Example
Bellman equation for Q-values

- Relationship between regular values and Q-values:

\[ V^*(s) = \max_a Q^*(s, a) \]

- Regular Bellman equation:

\[ V^*(s) = r(s) + \gamma \max_a \mathbb{E}_{s' \sim P(\cdot | s, a)} V^*(s') \]

- Bellman equation for Q-values:

\[
Q^*(s, a) = r(s) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} \max_{a'} Q^*(s', a') \\
= \mathbb{E}_{s' \sim P(\cdot | s, a)} [r(s) + \gamma \max_{a'} Q^*(s', a')] 
\]
Finding the optimal policy

• The Bellman equation is a constraint on Q-values of successive states:

\[
Q^*(s, a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} [r(s) + \gamma \max_{a'} Q^*(s', a')|s, a]
\]

• We could think of \(Q^*(s, a)\) as a table indexed by states and actions, and try to solve the system of Bellman equations to fill in the unknown values of the table

• **Problem:** state spaces for interesting problems are huge

• **Solution:** approximate Q-values using a parametric function:

\[
Q^*(s, a) \approx Q_w(s, a)
\]
RL: Outline

• Introduction to reinforcement learning
• Markov Decision Process (MDP) formalism and classical Bellman equation
• Q-learning
• Deep Q networks
Deep Q-learning

- Train a deep neural network to estimate Q-values:

Source: D. Silver

Deep Q-learning

\[ Q^*(s, a) = \mathbb{E}_{s' \sim p(\cdot | s, a)} [r(s) + \gamma \max_{a'} Q^*(s', a') | s, a] \]

- Idea: at each step of training, update model parameters \( w \) to “nudge” the left-hand side toward the right-hand “target”:

\[ y_{\text{target}}(s, a) = \mathbb{E}_{s' \sim p(\cdot | s, a)} [r(s) + \gamma \max_{a'} Q_{\text{target}}(s', a') | s, a] \]

- Loss function:

\[ L(w) = \mathbb{E}_{s, a} [(y_{\text{target}}(s, a) - Q_w(s, a))^2] \]
Deep Q-learning

• Target: \( y_{\text{target}}(s, a) = \mathbb{E}_{s' \sim P(\cdot|s, a)} [r(s) + \gamma \max_{a'} Q_{\text{target}}(s', a')|s, a] \)

• Loss: \( L(w) = \mathbb{E}_{s, a \sim \rho} [(y_{\text{target}}(s, a) - Q_w(s, a))^2] \)

• Gradient update:

\[
\nabla_w L(w) = \mathbb{E}_{s, a \sim \rho} [(y_{\text{target}}(s, a) - Q_w(s, a)) \nabla_w Q_w(s, a)]
\]

\[
= \mathbb{E}_{s, a \sim \rho, s'} [(r(s) + \gamma \max_{a'} Q_{\text{target}}(s', a') - Q_w(s, a)) \nabla_w Q_w(s, a)]
\]

• SGD training: replace expectation by sampling transitions \((s, a, s')\) using \textit{behavior distribution} and \textit{experience replay}
Deep Q-learning algorithm

- At each time step:
  - Take action $a_t$ according to *epsilon-greedy policy*
  - Store experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in *replay memory buffer*

<table>
<thead>
<tr>
<th>$s_1, a_1, r_2, s_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_2, a_2, r_3, s_3$</td>
</tr>
<tr>
<td>$s_3, a_3, r_4, s_4$</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>$s_t, a_t, r_{t+1}, s_{t+1}$</td>
</tr>
</tbody>
</table>

- Randomly sample *mini-batch* of experiences from the buffer
- Perform gradient descent step on loss:
  \[
  L(w) = \mathbb{E}_{s,a,s'} \left[ (r(s) + \gamma \max_{a'} Q_{\text{target}}(s', a') - Q_w(s, a))^2 \right]
  \]
- Update target network every $C$ steps
Deep Q-learning in Atari

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller,
Human-level control through deep reinforcement learning, Nature 2015
Deep Q-learning in Atari

- End-to-end learning of $Q(s, a)$ from pixels $s$
- Output is $Q(s, a)$ for 18 joystick/button configurations
- Reward is change in score for that step
Deep Q-learning in Atari

- Input state is stack of raw pixels (grayscale) from last 4 frames
- Network architecture and hyperparameters fixed for all games
Deep Q-learning in Atari
Breakout demo

https://www.youtube.com/watch?v=TmPfTpjtdgg
Outline

• Introduction to reinforcement learning
• Markov Decision Process (MDP) formalism and classical Bellman equation
• Q-learning
• Deep Q networks
• Extensions
  • Double DQN
  • Dueling DQN
Extension: Double Q-learning

- Max operator in standard Q-learning is used both to select and evaluate an action, leading to systematic over-estimation of Q-values
- Modification: select action using the online (current) network, but evaluate Q-value using the target network
- Regular DQN target:
  \[ y_{\text{target}}(s, a) = r(s) + \gamma \max_{a'} Q_{\text{target}}(s', a') \]
- Double DQN target:
  \[ y_{\text{target}}(s, a) = r(s) + \gamma Q_{\text{target}}(s', \text{argmax}_{a'} Q_w(s', a')) \]

Double DQN results
Another extension: Dueling DQN

- Decompose estimation of Q-function into value and advantage functions

Dueling DQN

- Decompose estimation of Q-function into *value* and *advantage* functions
  - Motivation: in many states, actions don’t meaningfully affect the environment, so it is not necessary to know the exact value of each action at each time step

Dueling DQN

- Decompose estimation of Q-function into value and advantage functions:

\[
Q(s, a) = V(s) + (A(s, a) - \max_{a'} A(s, a')) \quad \text{or}
\]

\[
Q(s, a) = V(s) + \left( A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a') \right)
\]

Dueling DQN: Results

Improvements over prioritized DDQN baseline: