Deep Q-learning
Reinforcement learning (RL)

- Agent can take actions that affect the state of the environment and observe occasional rewards that depend on the state.
- The goal is to 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
Applications of deep RL

- AlphaGo and AlphaZero

https://deepmind.com/research/alphago/
Applications of deep RL

• Playing video games

Applications of deep RL

• End-to-end training of deep visuomotor policies

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

Video
Sergey Levine et al., Berkeley
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

Transition model:

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

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 $r(s)$:

1. $R(s) < -1.6284$
2. $-0.4278 < R(s) < -0.0850$
3. $-0.0221 < R(s) < 0$
4. $R(s) > 0$
Rewards of state sequences

• Suppose that following policy \( \pi \) starting in state \( s_0 \) leads to a sequence \( s_0, s_1, s_2, \ldots \)

• The cumulative reward of the sequence is \( \sum_{t \geq 0} r(s_t) \)

• **Problem:** state sequences can vary in length or even be infinite

• **Solution:** redefine cumulative reward as sum of rewards *discounted* by a factor \( \gamma \):

\[
r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \gamma^3 r(s_3) + \cdots
= \sum_{t \geq 0} \gamma^t r(s_t), \quad 0 < \gamma \leq 1
\]
Discounting

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

- Cumulative reward is bounded by \( \frac{r_{\text{max}}}{1-\gamma} \)
- Helps algorithms converge

Image source: P. Abbeel and D. Klein
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 $s$:

$$V^\pi(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r(s_t) \mid s_0 = s, \pi \right]$$

with $a_t = \pi(s_t), s_{t+1} \sim P(\cdot \mid s_t, a_t)$

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

$$V^*(s) = \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r(s_t) \mid s_0 = s, \pi \right]$$
The Bellman equation

- Recursive relationship between optimal values of successive states:

\[ V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V^*(s') \]

Optimal policy:

\[ \pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)V^*(s') \]
The optimal policy

- Expression using the state value function:

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

- To use this in practice, we need to know the transition model

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

\[
Q^\pi(s, a) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r(s_t) \mid s_0 = s, a_0 = a, \pi \right]
\]
Q-value function

• The optimal Q-value:

\[ Q^*(s, a) = \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r(s_t) \mid s_0 = s, a_0 = a, \pi \right] \]

• What is the relationship between \( V^*(s) \) and \( Q^*(s, a) \)?

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

• What is the optimal policy?

\[ \pi^*(s) = \arg \max_a Q^*(s, a) \]
Q-value function

\[ Q^*(s, a) = \max_\pi \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r(s_t) \mid s_0 = s, a_0 = a, \pi \right] \]

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

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

- Regular Bellman equation:
  \[ V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s, a)V^*(s') \]

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

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

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

• 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:

\[ Q^*(s, a) \approx Q_w(s, a) \]
Deep Q-learning

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

\[
Q(s, a, w), \quad Q(s, a_1, w), \ldots, \quad Q(s, a_m, w)
\]

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 iteration \( i \) of training, update model parameters \( w_i \) to “nudge” the left-hand side toward the right-hand “target”
- Let \( y_i = \mathbb{E}_{s' \sim P(\cdot|s, a)} [r(s) + \gamma \max_{a'} Q_{w_{i-1}}(s', a')|s, a] \)
- The estimate \( Q_{w_i}(s, a) \) should match the target \( y_i \)
- Define loss function:
  \[ L_i(w_i) = \mathbb{E}_{s,a \sim \rho} [(y_i - Q_{w_i}(s, a))^2] \]
  where \( \rho \) is a behavior distribution
Deep Q-learning

• Target: \( y_i = \mathbb{E}_{s' \sim p(.|s,a)} [r(s) + \gamma \max_{a'} Q_{w_i-1}(s', a')|s, a] \)

• Loss: \( L_i(w_i) = \mathbb{E}_{s,a \sim \rho} [(y_i - Q_{w_i}(s, a))^2] \)

• Gradient update:

\[
\frac{\partial L}{\partial w_i} = \mathbb{E}_{s,a \sim \rho, s'} \left[ (y_i - Q_{w_i}(s, a)) \frac{\partial Q_{w_i}(s, a)}{\partial w_i} \right] = \mathbb{E}_{s,a \sim \rho, s'} \left[ (r(s) + \gamma \max_{a'} Q_{w_i-1}(s', a') - Q_{w_i}(s, a)) \frac{\partial Q_{w_i}(s, a)}{\partial w_i} \right]
\]

• SGD training: replace expectation by sampling experiences \((s, a, s')\) using behavior distribution and transition model
Deep Q-learning in practice

• Training is prone to instability
  • Unlike in supervised learning, the targets themselves are moving!
  • Successive experiences are correlated and dependent on the policy
  • Policy may change rapidly with slight changes to parameters, leading to drastic change in data distribution

• Solutions
  • Freeze target Q network
  • Use experience replay
Experience replay

- At each time step:
  - Take action $a_t$ according to \textit{epsilon-greedy policy}
  - Store experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in \textit{replay memory buffer}
  - Randomly sample \textit{mini-batch} of experiences from the buffer
Experience replay

• At each time step:
  • Take action $a_t$ according to \textit{epsilon-greedy policy}
  • Store experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in \textit{replay memory buffer}
  • Randomly sample \textit{mini-batch} of experiences from the buffer
  • Update parameters to reduce loss:

\[
L_i(w_i) = \mathbb{E}_{s,a,s'} \left[ (r(s) + \gamma \max_{a'} Q_{w_{i-1}}(s', a') - Q_{w_i}(s,a))^2 \right]
\]

Keep parameters of \textit{target network} fixed, update every once in a while
Deep Q learning in Atari

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