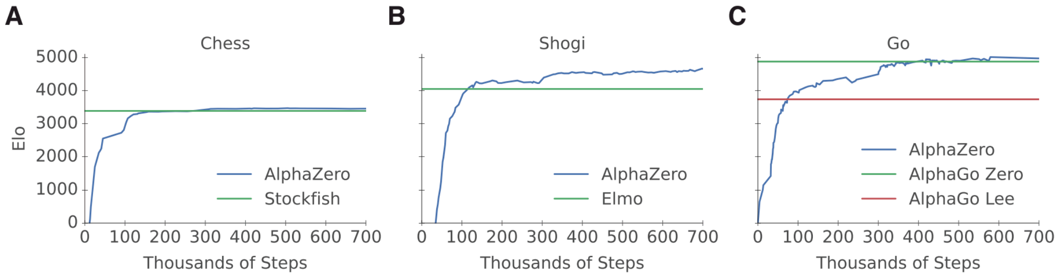
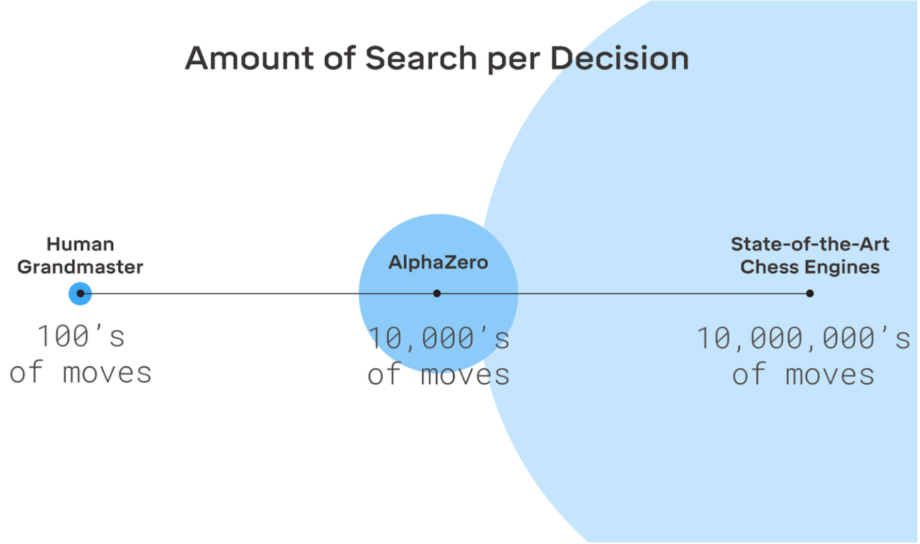
Reinforcement Learning

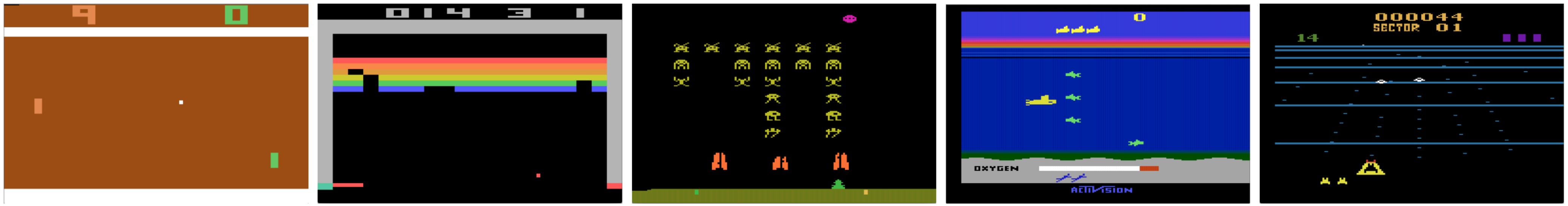
Tony Qin

Figures, and equations from David Silver https://www.davidsilver.uk/teaching/

Reinforcement Learning Applications



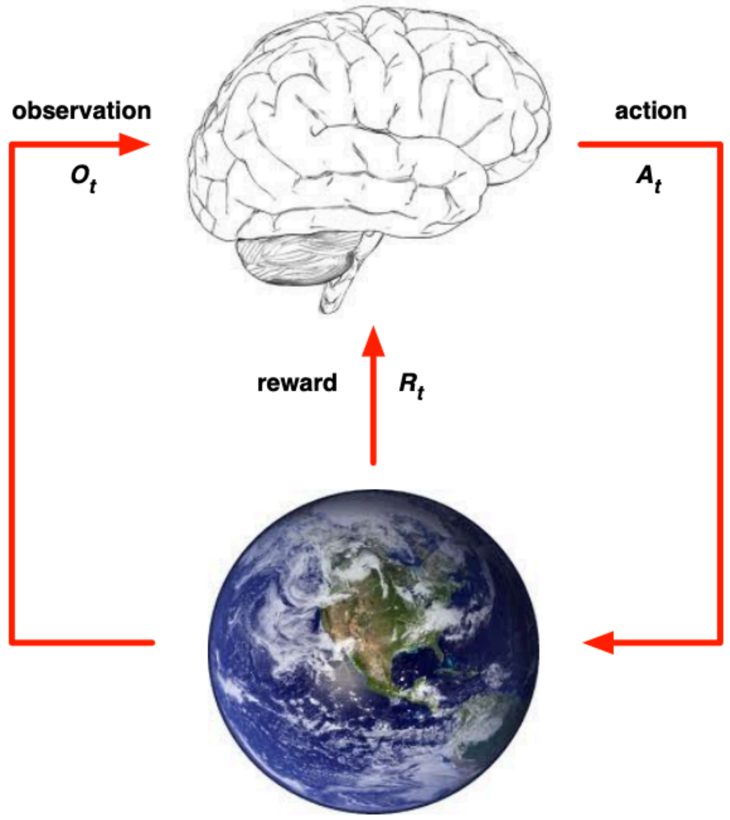




A general reinforcement learning algorithm that masters chess, shogi and Go through self-play, https://science.sciencemag.org/content/362/6419/1140.full?ijkey=XGd77kI6W4rSc&keytype=ref&siteid=sci
https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go
Playing Atari with Deep Reinforcement Learning, https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

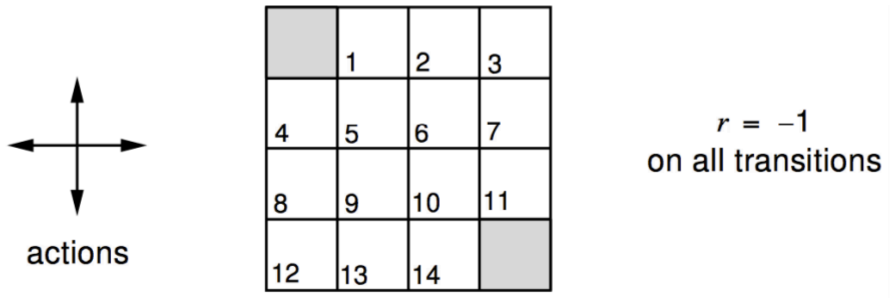
Agent and Environment

- Agent sees an observation O_t and reward R_t
- Agent takes an action A_t
- Environment responds to action A_t
- Environment emits observation O_{t+1} and reward R_{t+1}



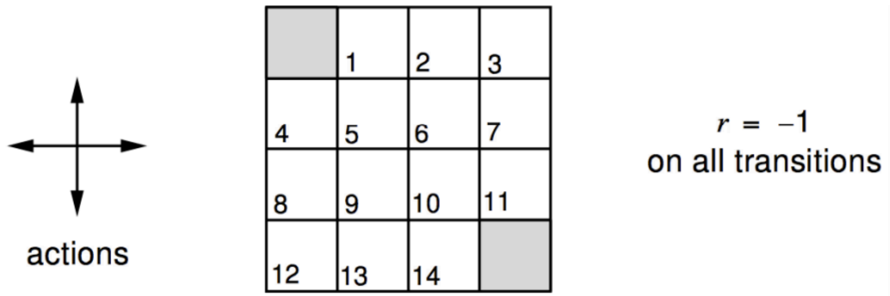
Markov Decision Process (MDP)

- S: set of finite states
- A: set of finite actions
- P: transition probability function
  - $P_{ss'}^{a} = P(S_{t+1} = s' | S_t = s, A_t = a)$
- R: reward function
  - $R_s^a = E(R_{t+1} | S_t = s, A_t = a)$
- γ: discount factor in $[0, 1]$
  - Return: $G_t = R_{t+1} + \gamma R_{t+2} + \ \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$



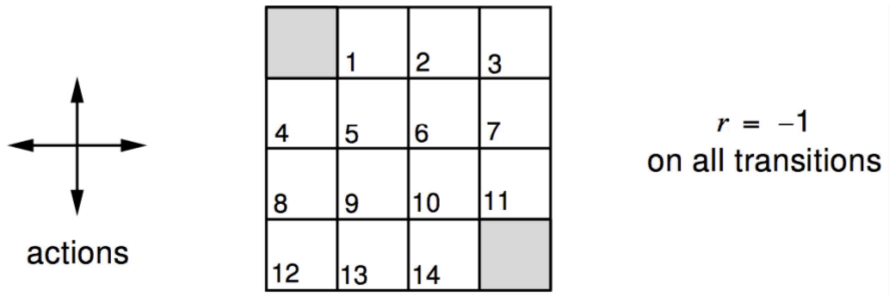
Markov Decision Process (MDP)

- S: set of finite states
- A: set of finite actions
- P: transition probability function
  - $P_{ss'}^{a} = P(S_{t+1} = s' | S_t = s, A_t = a)$
- R: reward function
  - $R_s^a = E(R_{t+1} | S_t = s, A_t = a)$
- γ: discount factor in $[0, 1]$
  - Return: $G_t = R_{t+1} + \gamma R_{t+2} + \ ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$



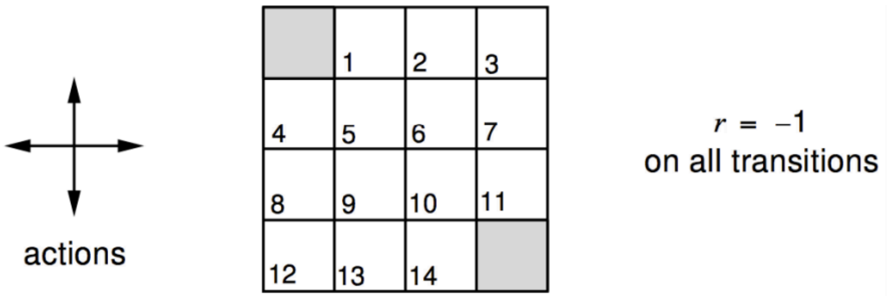
Markov Decision Process (MDP)

- S: set of finite states
- A: set of finite actions
- P: transition probability function
  - $P_{ss'}^{a} = P(S_{t+1} = s' | S_t = s, A_t = a)$
- R: reward function
  - $R_s^a = E(R_{t+1} | S_t = s, A_t = a)$
- γ: discount factor in $[0, 1]$
  - Return: $G_t = R_{t+1} + \gamma R_{t+2} + \ ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$



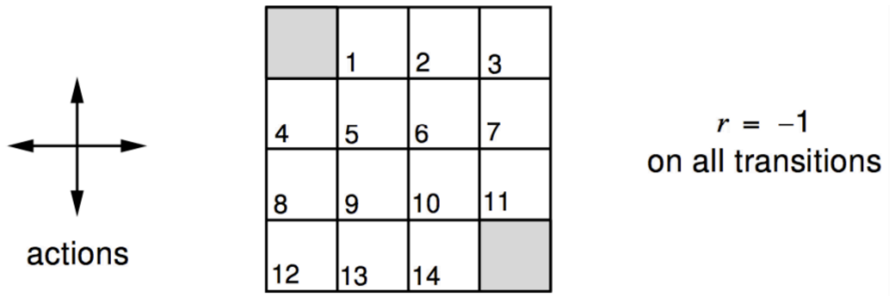
Markov Decision Process (MDP)

- S: set of finite states
- A: set of finite actions
- P: transition probability function
  - $P_{ss'}^{a} = P(S_{t+1} = s' | S_t = s, A_t = a)$
- R: reward function
  - $R_s^a = E(R_{t+1} | S_t = s, A_t = a)$
- γ: discount factor in $[0, 1]$
  - Return: $G_t = R_{t+1} + \gamma R_{t+2} + \ \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$



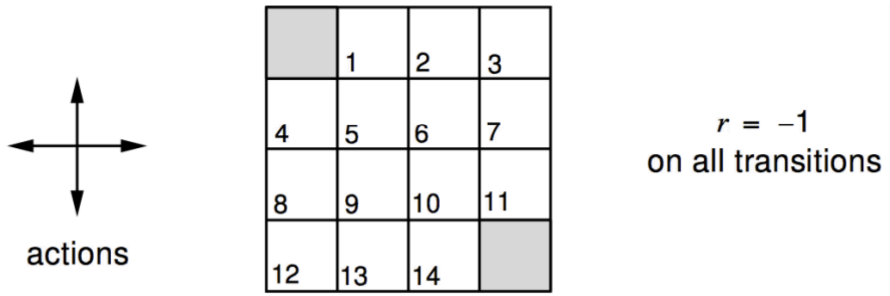
Markov Decision Process (MDP)

- S: set of finite states
- A: set of finite actions
- P: transition probability function
  - $P_{ss'}^{a} = P(S_{t+1} = s' | S_t = s, A_t = a)$
- R: reward function
  - $R_s^a = E(R_{t+1} | S_t = s, A_t = a)$
- γ: discount factor in $[0, 1]$
  - Return: $G_t = R_{t+1} + \gamma R_{t+2} + \ \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$



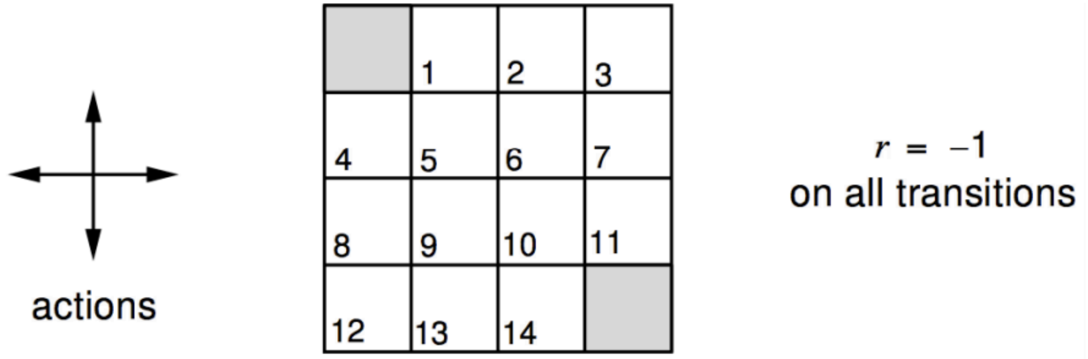
Markov Decision Process (MDP)

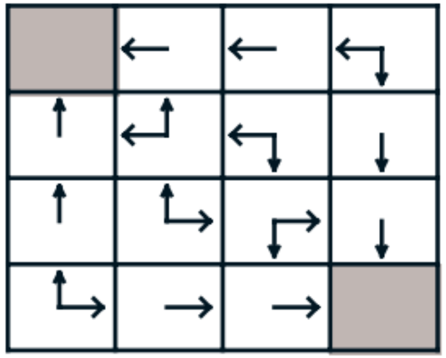
- S: set of finite states
- A: set of finite actions
- P: transition probability function
  - $P_{ss'}^{a} = P(S_{t+1} = s' | S_t = s, A_t = a)$
- R: reward function
  - $R_s^a = E(R_{t+1} | S_t = s, A_t = a)$
- γ: discount factor in $[0, 1]$
  - Return: $G_t = R_{t+1} + \gamma R_{t+2} + \ \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$



Value Functions

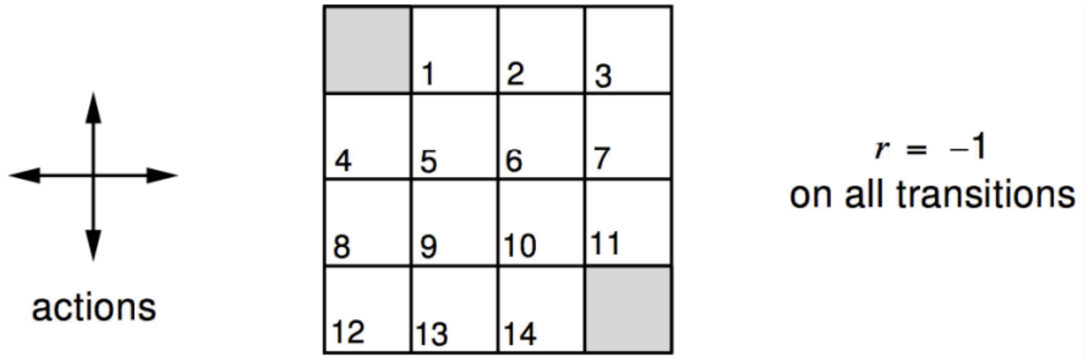
- Policy: $\pi(a|s) = \mathbb{P}(A_t = a \mid S_t = s)$
- Return: $G_t = R_{t+1} + \gamma R_{t+2} + \ \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
- State-value function: $v_\pi(s) = \ \mathbb{E}(G_t \mid S_t = s)$
- Action-value function: $q_\pi(s, a) = \mathbb{E}(G_t \mid S_t = s,\ A_t = a)$

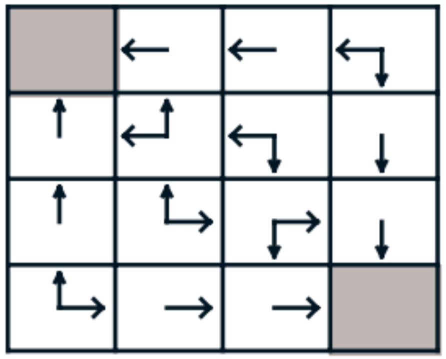




Optimal Value Functions

- There exists some optimal policy π^*
  - $v_\pi(s) \geq v_{\pi'}(s), \forall s$
- Optimal state-value function $v_*(s) = \max_\pi v_\pi(s)$
  - $v_{\pi_*}(s) = v_*(s)$
- Optimal action-value function $q_*(s,a) = \max_\pi q_\pi(s,a)$
  - $q_{\pi_*}(s,a) = q_*(s,a)$





Value Iteration

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \boldsymbol{\mathcal{R}}^{\boldsymbol{a}} + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{a}} \mathbf{v}_k$$

$$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \; \ldots \rightarrow v_*$$

Converges to v_*

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Value Iteration

$$v_{k+1}(s) = \max_{a\in\mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}_{k+1} = \max_{a\in\mathcal{A}} \boldsymbol{\mathcal{R}}^{\mathbf{a}} + \gamma \boldsymbol{\mathcal{P}}^{\mathbf{a}} \mathbf{v}_k$$

$$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \ \dots \rightarrow v_*$$

Converges to v_*

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Value Iteration

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \boldsymbol{\mathcal{R}}^{\mathbf{a}} + \gamma \boldsymbol{\mathcal{P}}^{\mathbf{a}} \mathbf{v}_k$$

$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \ \ldots \rightarrow v_*$

Converges to v_*

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

SARSA

- Model free: don't know transition and reward function
- Can't use value iteration
- $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$

```
Initialize Q(s,a), ∀s ∈ S, a ∈ A(s), arbitrarily, and Q(terminal-state, ·) = 0
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        Q(S,A) ← Q(S,A) + α[R + γQ(S',A') − Q(S,A)]
        S ← S'; A ← A';
    until S is terminal
```

Q Learning

- Off policy: learn from episodes generated with a different policy
- $Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma \max_{a'} Q(S',a') - Q(S,A))$

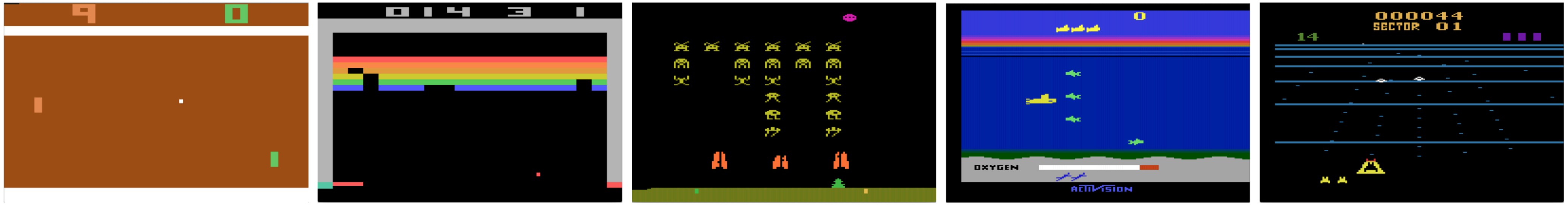
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$;
    until S is terminal

Value Function Approximation

- The previous methods are resource intensive
  - Storing values requires $O(|S|)$ memory
- Intractable for problems with large state spaces
  - Go: 10^{170} states
  - Robotics: continuous state space
- Use neural networks to approximate value functions
  - $\hat{v}(s, \theta) \approx v_\pi(s)$
  - $\hat{q}(s, a, \theta) \approx q_\pi(s, a)$

DQN

- Store $(s_t, a_t, r_{t+1}, s_{t+1})$ tuples in replay memory D
- $\mathrm{L} = \mathbb{E}_{s,a,r,s' \sim D}\left(R + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a, \theta)\right)$
- Sample batch of transitions from memory
- Used in famous paper to play Atari games



Playing Atari with Deep Reinforcement Learning, https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

DQN

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

Playing Atari with Deep Reinforcement Learning, https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

Conclusion

- Check out https://www.davidsilver.uk/teaching/
- Key papers in RL: https://spinningup.openai.com/en/latest/spinningup/keypapers.html
- Have a good winter break