RL Algorithms: Q-learning

Model-Free vs. Model-Based RL

- Bellman equation (state-action presentation):

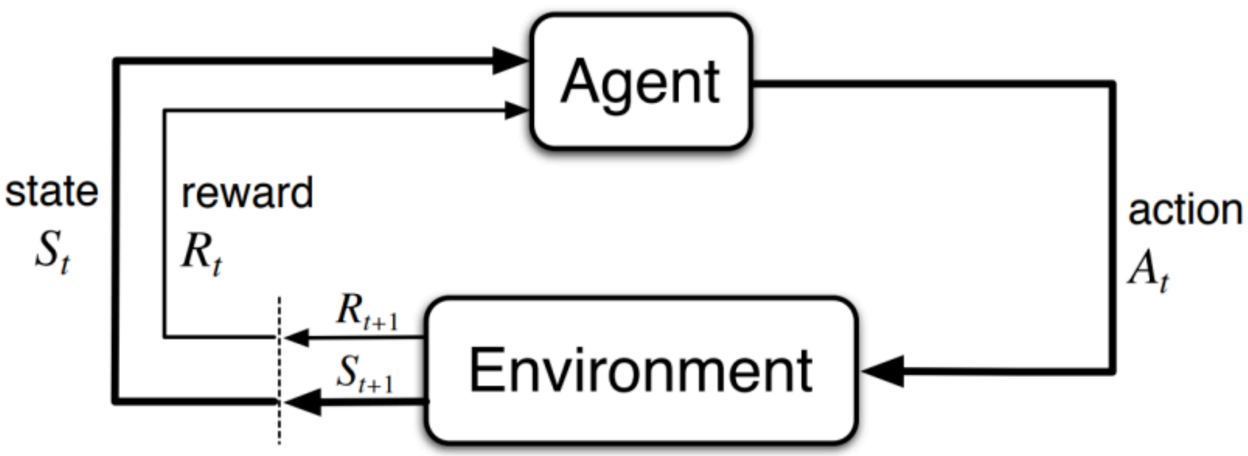
$$
\begin{aligned}
Q^*(s,a) &= E(r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,|\, s_t = s, a_t = a) \\
&= \sum_{s'} P^a_{ss'}(R^a_{ss'} + \gamma \max_{a'} Q^*(s', a'))
\end{aligned}
$$

- Explicit knowledge of transition probabilities not needed to learn the optimal policy

- **Model free** vs. **model based** RL

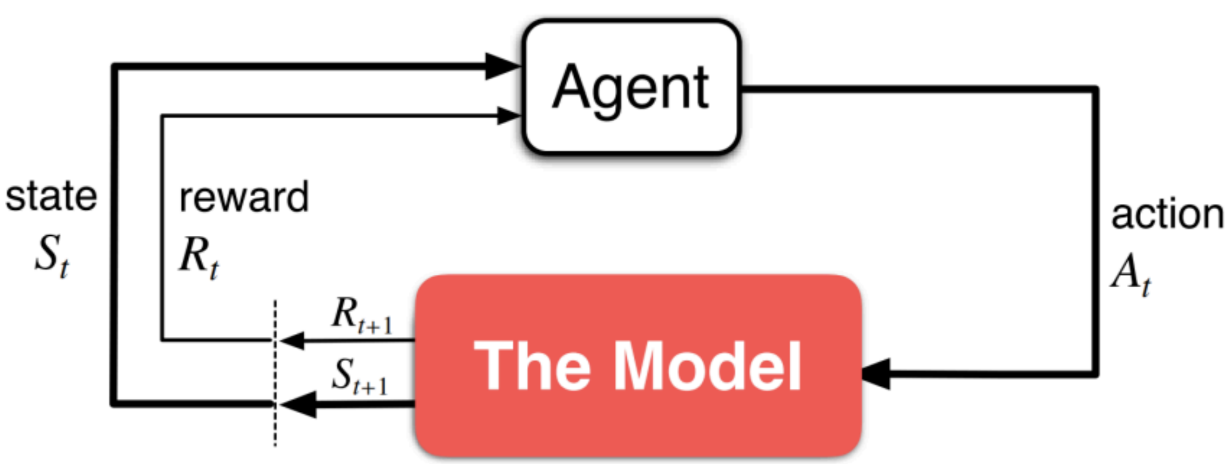
- Model free RL - learn in environment or the simulator!

Model-Free vs Model-Based RL



- Environment is noisy
- Takes longer to converge to optimal policy

$$\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}, \quad \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$$

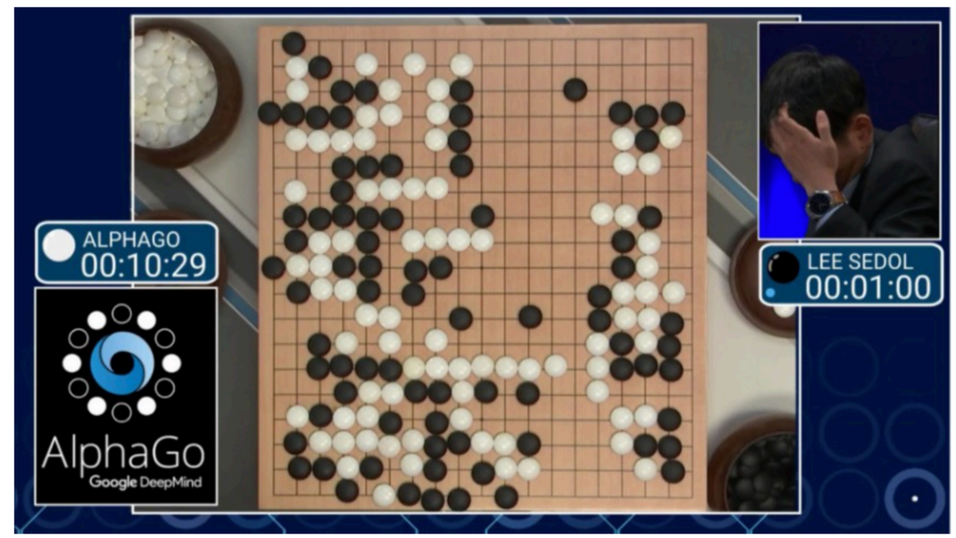


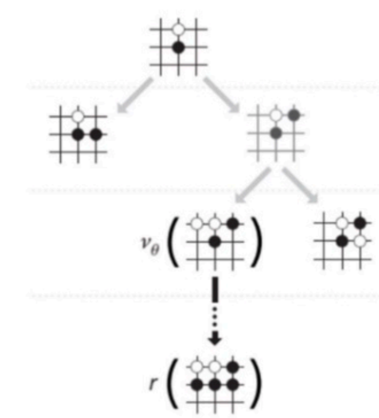
- Need to learn the model
- Learning optimal policy is **sample-efficient**
- Optimal policy is as good as the model

$$\underbrace{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}_{\text{model}}, \quad \cancel{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}$$

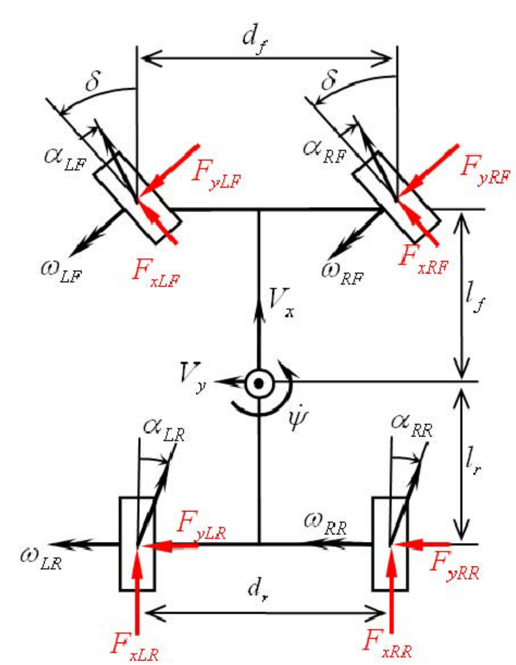
Model-based examples

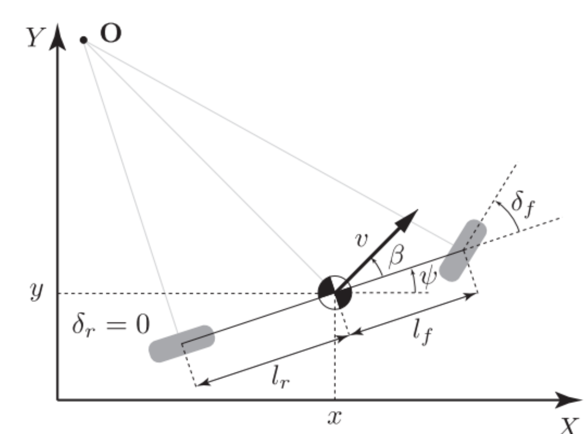
- Frog-escape
- Games: go or chess (can also be model-free)
- Physical systems



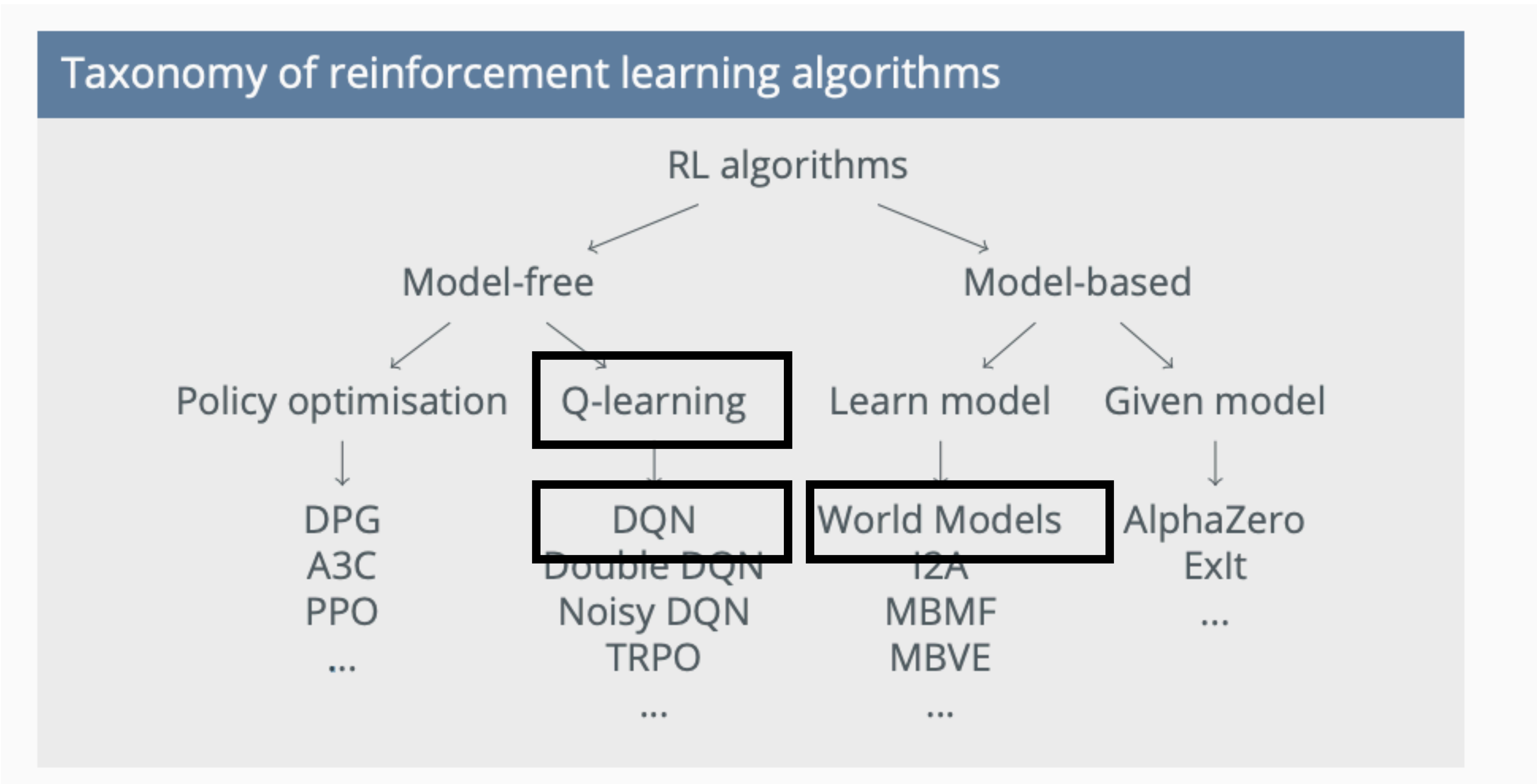


AlphaGO





Taxonomy of RL algorithms



Ideas for capstone projects!

Tabular Q-Learning

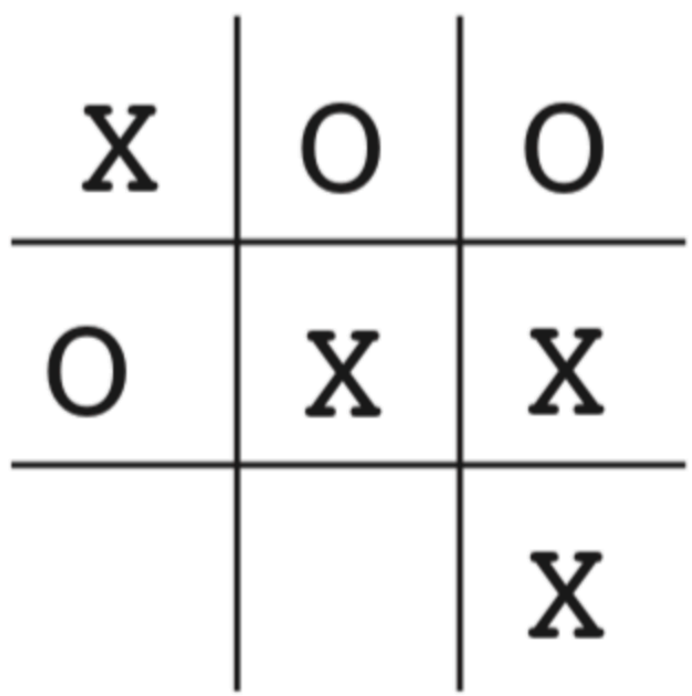
$$Q^*(s,a) = \qquad E(r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,|\, s_t = s, a_t = a)$$

- Q - “quality”

- Q function is the function that assigns a quality score to an State — Action pair (Action Value Function)

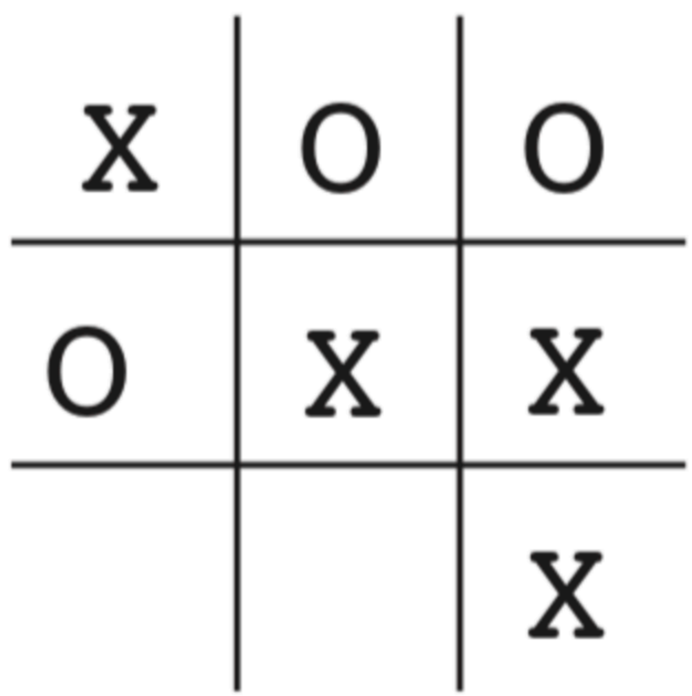
- Given a state *s* and an action *a* the function *Q(s,a)* will return a real number reflecting the quality of doing this action *a* in the state *s*

Tic Tac Toe

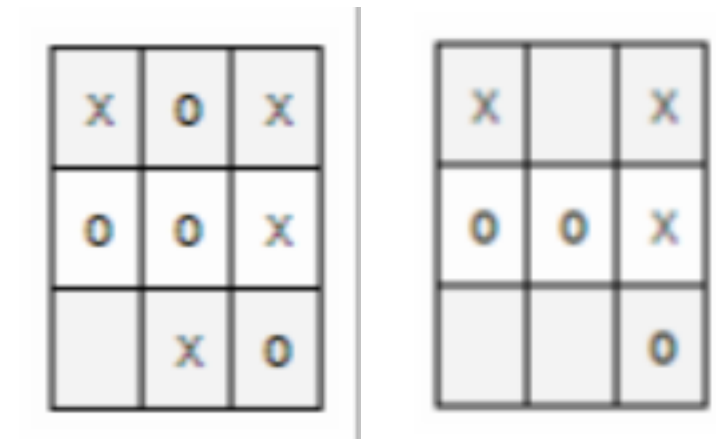


- If played against an **optimal opponent** many times, can learn the best strategy
- **States:**
- **Actions:**
- **Transition probabilities:**
- **Rerwards:**

Tic Tac Toe



- **States:**



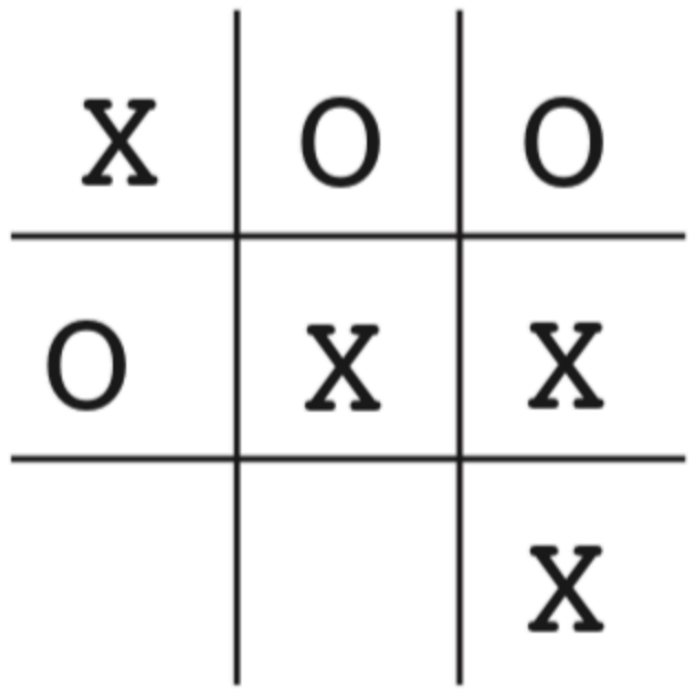
- **Actions:**

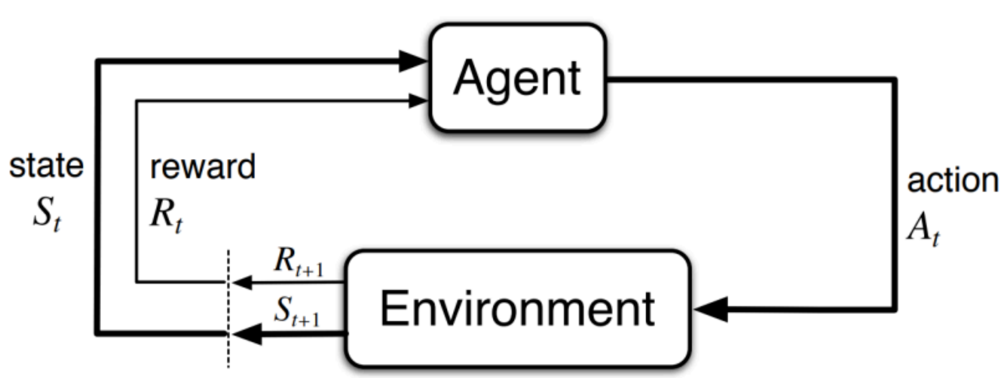
Top Left	Top Middle	Top Right	Middle Left	Middle Middle	Middle Right	Bottom Left	Bottom Middle	Bottom Right

- **Transition probabilities: implied by the game**

- **Rewards: 1 for winning the game, 0.5 for a tie, 0 for loosing a game**

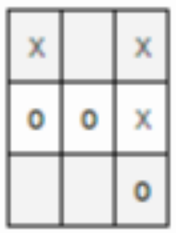
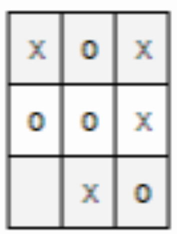
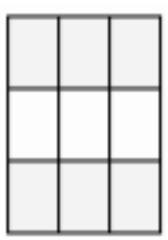
Tic Tac Toe Reward Assignment





- Rewards should **motivate** the agent to learn to take best action at a state
- Good action - positive reward
- Bad action - negative reward
- Tic tac toe rewards at the end of the game
- Need to attribute past rewards back to previous actions

Q-Table

Game State	Top Left	Top Middle	Top Right	Middle Left	Middle Middle	Middle Right	Bottom Left	Bottom Middle	Bottom Right
	N/A	0.5	N/A	N/A	N/A	N/A	0	0	N/A
	N/A	N/A	N/A	N/A	N/A	N/A	0.5	N/A	N/A
	0.3	0.5	0.3	0.5	0.7	0.3	0.3	0.5	0.3
...	...	...	...	...	...	...	...	...	...

Q-Learning Algorithm

$$Q^*(s,a) = \qquad E(r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \,|\, s_t = s, a_t = a)$$

- Initialize the Q-table
- Until convergence:

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{current value}} \bigg)}^{\text{temporal difference}}$$

Hyperparamers of Q-Learning

- Learning rate $0 \le \alpha < 1$
  - For convergence, learning rate must decrease to 0
- Discount factor $0 \le \gamma < 1$
  - Importance of future reward

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{current value}} \right)}^{\text{temporal difference}}$$

Q-Table in Training

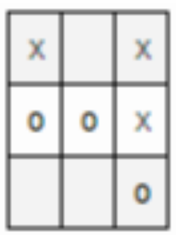
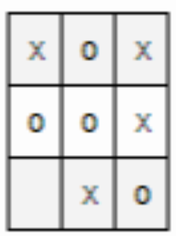
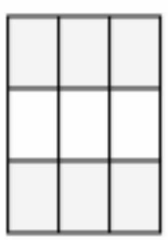
Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	327	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

From Q-Table to Optimal Policy

Game State	Top Left	Top Middle	Top Right	Middle Left	Middle Middle	Middle Right	Bottom Left	Bottom Middle	Bottom Right
	N/A	0.5	N/A	N/A	N/A	N/A	0	0	N/A
	N/A	N/A	N/A	N/A	N/A	N/A	0.5	N/A	N/A
	0.3	0.5	0.3	0.5	0.7	0.3	0.3	0.5	0.3
...	...	...	...	...	...	...	...	...	...

Convergence of Q-Learning

- Tabular Q-learning is guaranteed to converge to a globally optimal policy when all states are visited infinitely many times the below conditions are satisfied

$$\sum_{i=0}^{\infty} \alpha_{i(s,a)} = \infty, \quad \sum_{i=0}^{\infty} \alpha_{i(s,a)}^2 < \infty, \quad \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{10}$$

- Tabular Q - states need to be discretized, not always scalable

- Q-function can be learned in different shapes (e.g., DQN), but the convergence results are generally not guaranteed

Exploration Q-Learning

- Tabular Q-learning is guaranteed to converge to a globally optimal policy when **all states are visited infinitely many times** the below conditions are satisfied

$$\sum_{i=0}^{\infty} \alpha_{i(s,a)} = \infty, \quad \sum_{i=0}^{\infty} \alpha_{i(s,a)}^2 < \infty, \quad \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{10}$$

- $\epsilon -$ **greedy**
  - Take random action at a state with probability ϵ

Rewards

- Q-learning agent is trained to maximizes total cumulative rewards