Deep learning

Deep reinforcement learning

Hamid Beigy

Sharif University of Technology

December 23, 2024

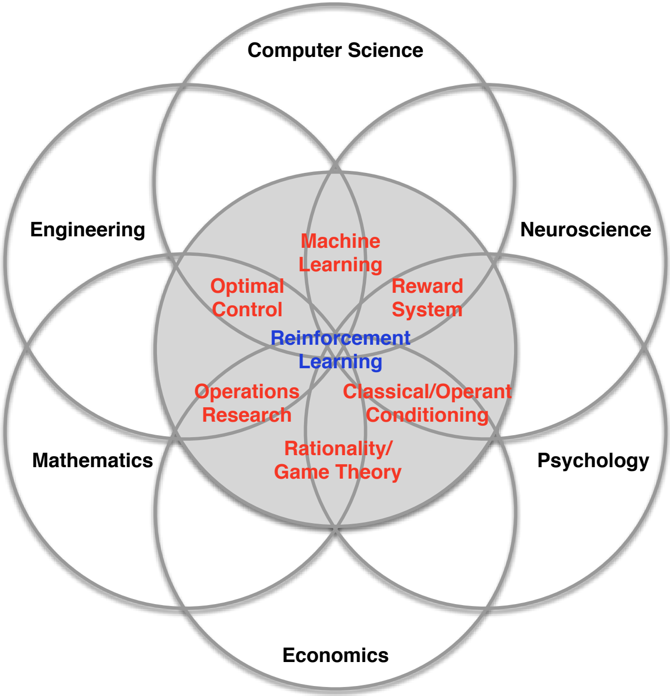
Table of contents

Introduction

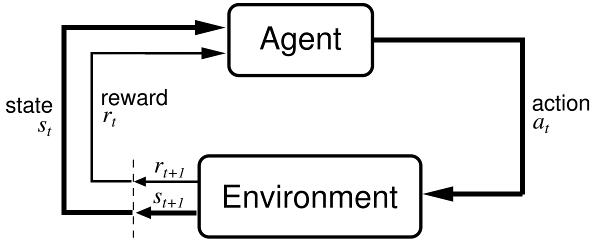
Introduction

1. Reinforcement learning is what to do (how to map situations to actions) so as to maximize a scalar reward/reinforcement signal
2. The learner is not told which actions to take as in supervised learning, but discover which actions yield the most reward by trying them.
3. The trial-and-error and delayed reward are the two most important feature of reinforcement learning.
4. Reinforcement learning is defined not by characterizing learning algorithms, but by characterizing a learning problem.
5. Any algorithm that is well suited for solving the given problem, we consider to be a reinforcement learning.
6. One of the challenges that arises in reinforcement learning and other kinds of learning is tradeoff between exploration and exploitation.

Introduction (Faces of RL)



1. A key feature of reinforcement learning is that it explicitly considers the whole problem of a goal-directed agent interacting with an uncertain environment.



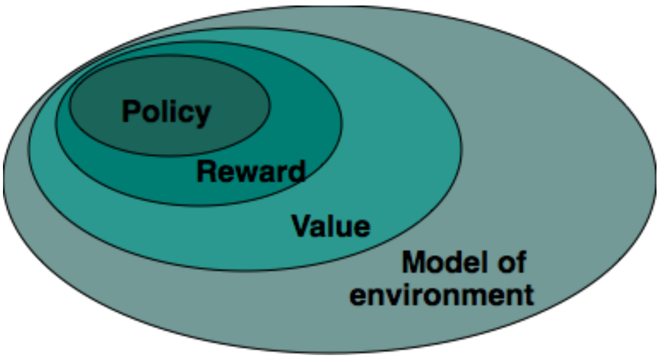
2. Experience is a sequence of observations, actions, rewards: $o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t$
3. The state is a summary of experience : $s_t = f(o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t)$
4. In a fully observed environment : $s_t = f(o_t)$

Policy A policy is a mapping from received states of the environment to actions to be taken (what to do?).

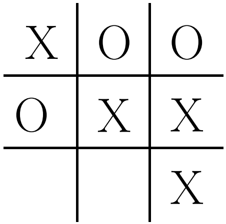
Reward function It defines the goal of RL problem. It maps each state-action pair to a single number called reinforcement signal, indicating the goodness of the action. (what is good?)

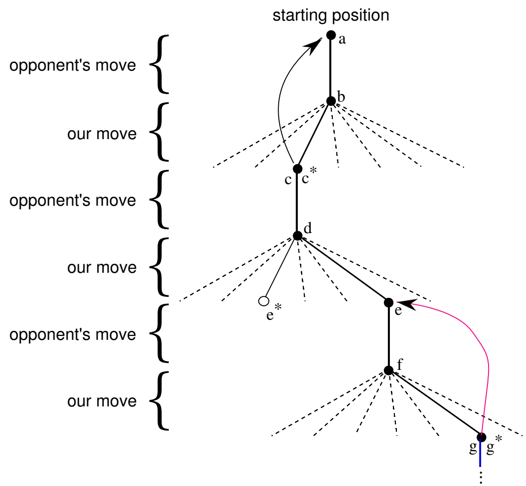
Value It specifies what is good in the long run. (what is good because it predicts reward?)

Model of the environment This is something that mimics the behavior of the environment. (what follows what?) This element is optional.



1. Consider a two-playes game (Tic-Tac-Toe)

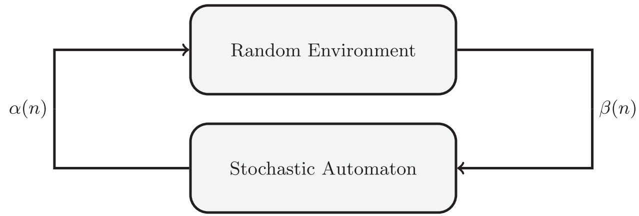




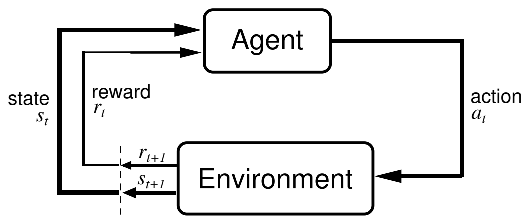
2. Consider the following updating

$$V(s) \leftarrow V(s) + \alpha[V(s') - V(s)]$$

Non-associative reinforcement learning The learning method that does not involve learning to act in more than one state.



Associative reinforcement learning The learning method that involves learning to act in more than one state.



Multi-arm Bandit problem

1. Consider that you are faced repeatedly with a choice among n different options or actions.
2. Then receive a numerical reward that depends on the selected action.
3. The numerical reward is chosen from a stationary probability distribution.
4. The objective is to maximize the expected total reward over some time period.
5. This is the original form of n-armed bandit problem called slot machine.

Action-value methods

1. Consider some simple methods for estimating the values of actions and then using the estimates to select actions.
2. Let true value of action a denoted as $Q^*(a)$ and its estimated value at t^{th} play as $Q_t(a)$.
3. The true value of an action is the mean reward when that action is selected.
4. One natural way to estimate this is by averaging the rewards actually received when the action was selected.
5. In other words, if at the t^{th} play action a has been chosen k_a times prior to t, yielding rewards $r_1, r_2, \ldots, r_{k_a}$, then its value is estimated to be

$$Q_t(a) = \frac{r_1 + r_2 + \ldots + r_{k_a}}{k_a}$$

1. **Greedy action selection:** This strategy selects the action with highest estimated action value.

$$a_t = \underset{a}{\operatorname{argmax}}\ Q_t(a)$$

2. $\epsilon-$**greedy action selection:** This strategy selects the action with highest estimated action value most of time but with small probability ϵ selects an action at random, uniformly, independently of the action-value estimates.
3. **Softmax action selection:** This strategy selects actions using the action probabilities as a graded function of estimated value.

$$p_t(a) = \frac{\exp^{Q_t(a)/\tau}}{\sum_b \exp^{Q_t(b)/\tau}}$$

Goals, rewards, and returns

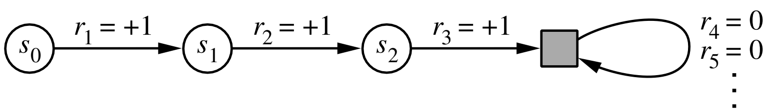
1. In RL, the goal of agent is defined in terms of a reward from environment to agent.
2. The agent's goal is to **maximize total amount of reward** it receives.
3. This means maximizing not **immediate reward**, but **cumulative reward** in the long run.
4. How might the goal be formally defined?
   - In **episodic tasks** the return, R_t, is defined as

$$R_t = r_1 + r_2 + \ldots + r_T$$

   - In **continuous tasks** the return, R_t, is defined as

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

5. The **unified approach**

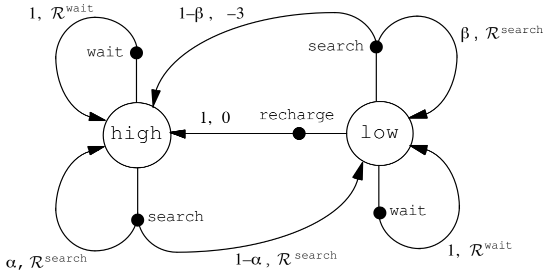


Markov decision process

1. A RL task satisfying the **Markov property** is called a Markov decision process (MDP).
2. If state and action spaces are finite, then it is called a finite MDP.
3. A finite MDP is defined by its state / action sets and one-step dynamics of environment.

$$\mathcal{P}^a_{ss'} = \mathbb{P}\left[s_{t+1} = s' | s_t = s, a_t = a\right]$$
$$\mathcal{R}^a_{ss'} = \hat{\mathbb{E}}\left[r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\right]$$

4. Recycling Robot MDP



1. Let in state s, action a is selected with probability of $\pi(s, a)$.
2. Value of s under policy π is the expected return when starting in s and following π thereafter.

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\pi}[R_t \mid s_t = s] = \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s\right]$$
$$= \sum_{\pi} \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V^{\pi}(s')\right].$$

3. Value of a in state s under policy π is the expected return when starting in s taking action a and following π thereafter.

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\pi}[R_t \mid s_t = s, a_t = a] = \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a\right]$$

Optimal value functions

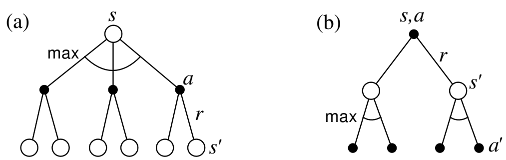
1. Policy π is better than or equal of π' iff for all s, we have $V^\pi(s) \geq V^{\pi'}(s)$.
2. There is always at least one policy that is better than or equal to all other policies. This is an optimal policy.
3. Value of state s under the optimal policy ($V^*(s)$) equals

$$V^*(s) = \max_\pi V^\pi(s)$$

4. Value of action a in state s under the optimal policy ($Q^*(s, a)$ equals

$$Q^*(s, a) = \max_\pi Q^\pi(s, a)$$

5. Backup diagram for V^* and Q^*



Approaches to RL

1. Model-based RL
   - Build a model of the environment.
   - Plan (e.g. by lookahead) using model.
2. Value-based RL
   - Estimate the optimal value function $Q^*(s, a)$
   - This is the maximum value achievable under any policy
3. Policy-based RL
   - Search directly for the optimal policy π^*.
   - This is the policy achieving maximum future reward.

Model based methods

Model based methods (dynamic programming)

1. The key idea of DP is the use of value functions to organize and structure the search for good policies.
2. We can easily obtain optimal policies once we have found the optimal value functions, or , which satisfy the Bellman optimality equations:

$$\begin{aligned} V^*(s) &= \max_a \hat{\mathbb{E}} \left[r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a \right] \\ &= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^*(s') \right]. \end{aligned}$$

3. Value of action a in state s under a policy π is the expected return when starting in s taking action a and following π thereafter.

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

Policy iteration

1. Policy iteration is an iterative process

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots\ldots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

2. Policy iteration has two phases : policy evaluation and policy improvement.
3. In policy evaluation, we compute state or state-action value functions

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi [R_t \mid s_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right] \\ &= \sum_\pi \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s') \right]. \end{aligned}$$

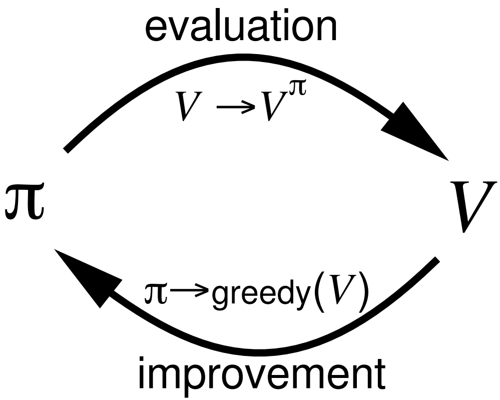
4. In policy improvement, we change the policy to obtain a better policy

$$\begin{aligned} \pi'(s) &= \underset{a}{\operatorname{argmax}}\ Q^\pi(s,a) \\ &= \underset{a}{\operatorname{argmax}} \sum_{s'} P^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s') \right]. \end{aligned}$$

1. In value iteration we have

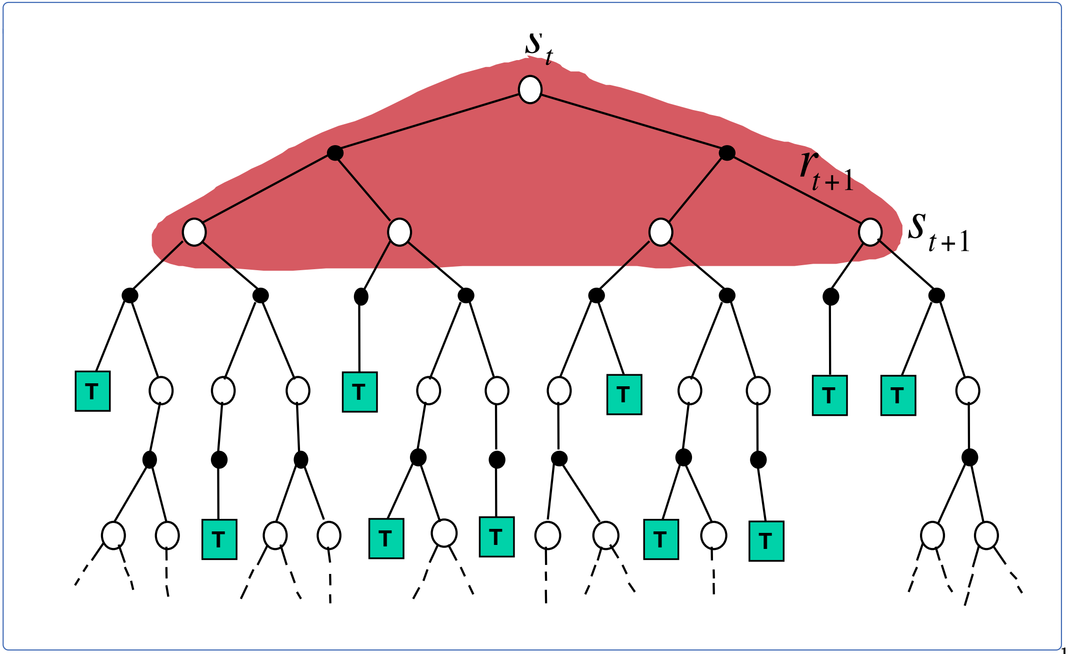
$$\begin{aligned} V_{k+1}(s) &= \max_a \mathbb{E}\left[r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s, a_t = a\right] \\ &= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k^(s')\right]. \end{aligned}$$

2. Generalized policy iteration



DP Backup diagram

$$V(S_t) \leftarrow \hat{\mathbb{E}}_{\pi}[R_{t+1} + \gamma V(S_{t+1})]$$



Value-based methods

Value-based methods

1. These methods lean policy function implicitly.
2. These methods first learn a value function $Q(s, a)$.
3. Then infer policy $\pi(s, a)$ from $Q(s, a)$.
4. Examples
   - Monte-carlo methods
   - Temporal difference methods
     - Q-learning
     - SARSA
     - TD(λ)
     - ...

Value-based methods

Monte Carlo methods

1. MC methods learn directly from episodes of experience.
2. MC is model-free: no knowledge of MDP transitions / rewards
3. MC learns from complete episodes
4. MC uses the simplest possible idea: value = mean return
5. Goal: learn V_π from episodes of experience under policy π

$$S_1 \xrightarrow[R_1]{\alpha_1} S_2 \xrightarrow[R_2]{\alpha_2} S_3 \xrightarrow[R_3]{\alpha_3} S_4 \ldots \xrightarrow[R_{k-1}]{\alpha_{k-1}} S_k$$

6. The return is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$$

7. The value function is the expected return:

$$V_\pi(s) = \mathop{\mathbb{E}}_{\pi}[G_t \mid S_t = s]$$

8. Monte-Carlo policy evaluation uses empirical mean return instead of expected return

First-Visit Monte-Carlo Policy Evaluation

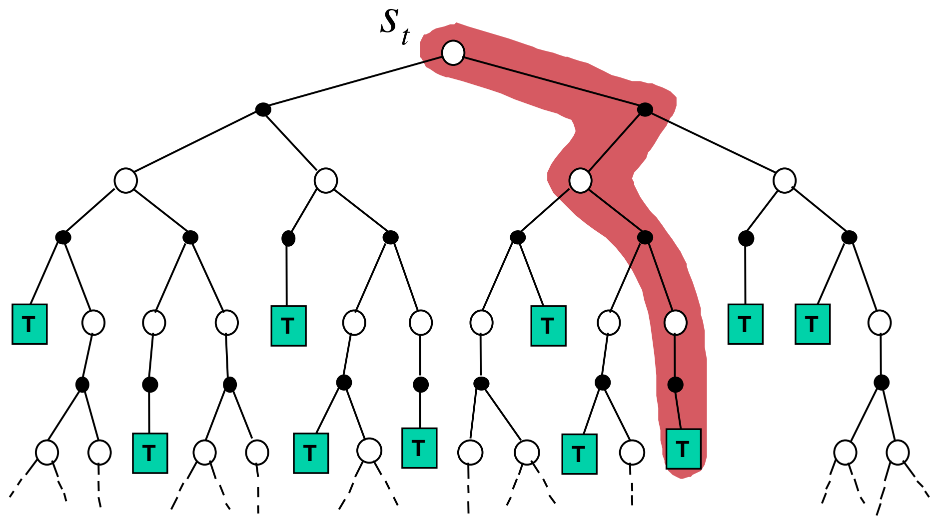
1. To evaluate state s
2. The first time-step t that state s is visited in an episode, Increment counter
$$N(s) \leftarrow N(s) + 1$$
3. Increment total return
$$S(s) \leftarrow S(s) + G_t$$
4. Value is estimated by mean return
$$V(s) = \frac{S(s)}{N(s)}$$
5. By law of large numbers,
$$V(s) \rightarrow V_\pi(s)$$
as
$$N(s) \rightarrow \infty$$

Every-Visit Monte-Carlo Policy Evaluation

1. To evaluate state s
2. Every time-step t that state s is visited in an episode, Increment counter
$$N(s) \leftarrow N(s) + 1$$
3. Increment total return
$$S(s) \leftarrow S(s) + G_t$$
4. Value is estimated by mean return
$$V(s) = \frac{S(s)}{N(s)}$$
5. By law of large numbers,
$$V(s) \to V_\pi(s)$$
as
$$N(s) \to \infty$$

MC Backup diagram

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$



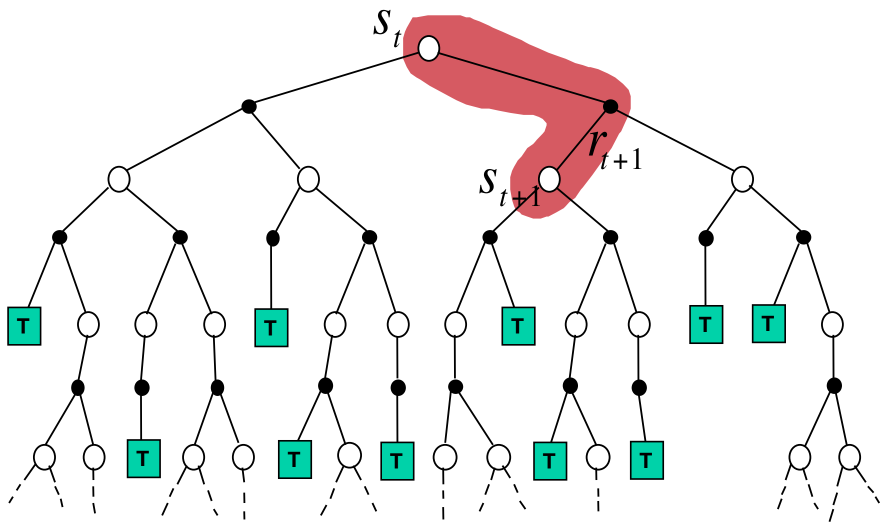
Value-based methods

Temporal-difference methods

1. TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas.
2. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics.
3. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap).
4. Monte Carlo methods wait until the return following the visit is known, then use that return as a target for $V(s_t)$ while TD methods need wait only until the next time step.
5. The simplest TD method, known as TD(0), is

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

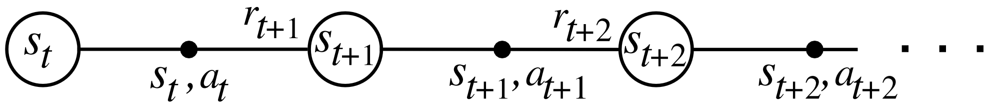


1. Algorithm for TD(0)

Initialize $V(s)$ arbitrarily, π to the policy to be evaluated
Repeat (for each episode):
. Initialize s
. Repeat (for each step of episode):
. . $a \leftarrow$ action given by π for s
. . Take action a; observe reward, r, and next state, s'
. . $V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$
. . $s \leftarrow s'$
. until s is terminal

---

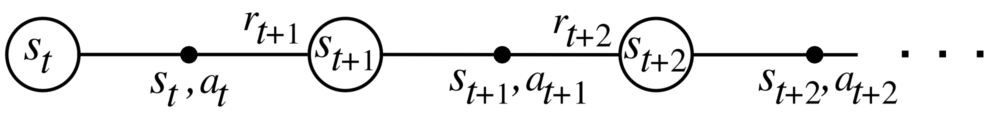
1. An episode consists of an alternating sequence of states and state-action pairs:



2. SARSA, which is an on policy, updates values using

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

1. An episode consists of an alternating sequence of states and state-action pairs:



2. Q-learning, which is an off policy, updates values using

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Policy-based methods

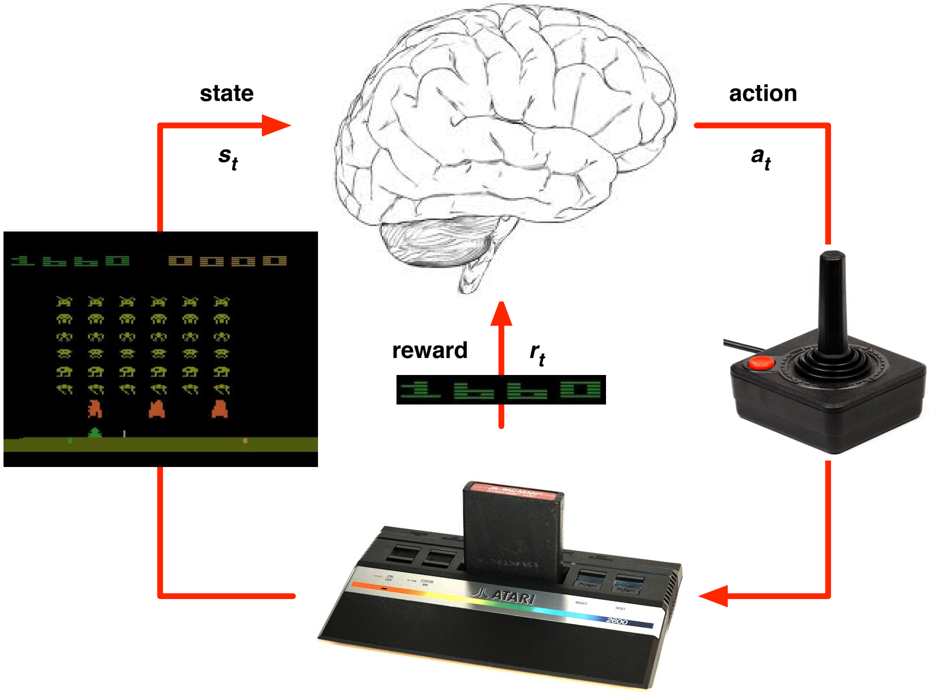
Policy-based methods

1. In policy-based learning, there is no value function.
2. The policy $\pi(s, a)$ is parametrized by vector θ ($\pi(s, a; \theta)$).
3. Explicitly learn policy $\pi(s, a; \theta)$ that implicitly maximize reward over all policies.
4. Given policy $\pi(s, a; \theta)$ with parameters θ, find best θ.
5. How do we measure the quality of a policy $\pi(s, a; \theta)$?
6. Let objective function be $J(\theta)$.
7. Find policy parameters θ that maximize $J(\theta)$.
8. Sample algorithm: REINFORCE

Policy-based methods versus value-based methods

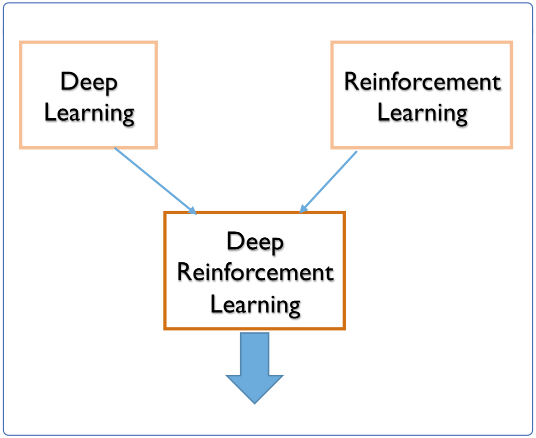
1. Advantages of policy-based methods over value-based methods
   - Usually, computing Q-values is harder than picking optimal actions
   - Better convergence properties
   - Effective in high dimensional or continuous action spaces
   - Can benefit from demonstrations
   - Policy subspace can be chosen according to the task
   - Exploration can be directly controlled
   - Can learn stochastic policies
2. Disadvantages of policy-based methods over value-based methods
   - Typically converge to a local optimum rather than a global optimum
   - Evaluating a policy is typically data inefficient and high variance

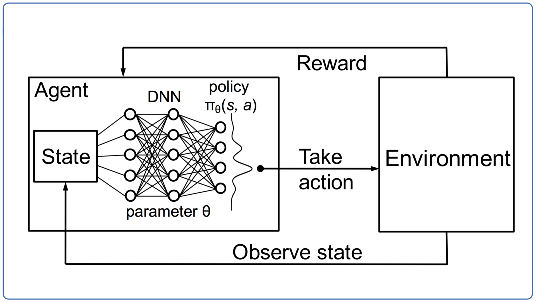
Deep reinforcement learning


state
s_t
action
a_t
reward
r_t
ATARI
2600

Deep Reinforcement Learning

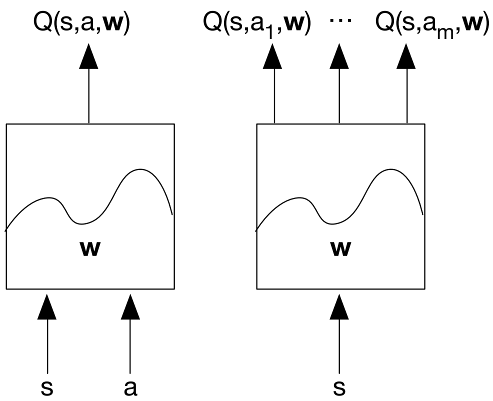
1. Use deep network to represent value function/ policy/model.
2. Optimize value function/ policy/model end–to–end.
3. Use stochastic gradient descent.





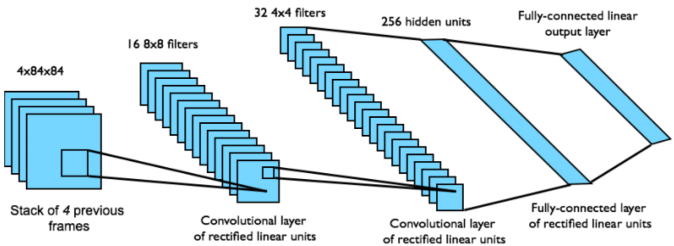
Value-Based Deep RL

1. Represent value function by Q-network with weights w : $Q(s, a; w) \approx Q^*(s, a)$

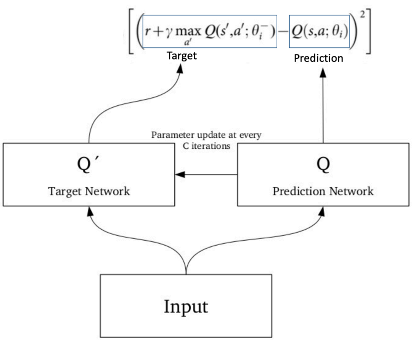


Deep Q-Network (Mnih et al. 2015)

1. End-to-end learning of values $Q(s, a)$ from pixels s.
2. Input state s is stack of raw pixels from last 4 frames
3. Output is $Q(s, a)$ for 18 joystick/button positions
4. Reward is change in score for that step



1. Deep Q-network consists of
   - Q network predicting Q-values
   - Target network, which has the same structure as Q-network
   - Experience replay component



2. Experience replay selects an $\epsilon-$greedy action from the current state, executes it in the environment, and gets back a reward and the next state. It saves this observation as a sample of training data.

Deep Q-Network (Mnih et al. 2015)

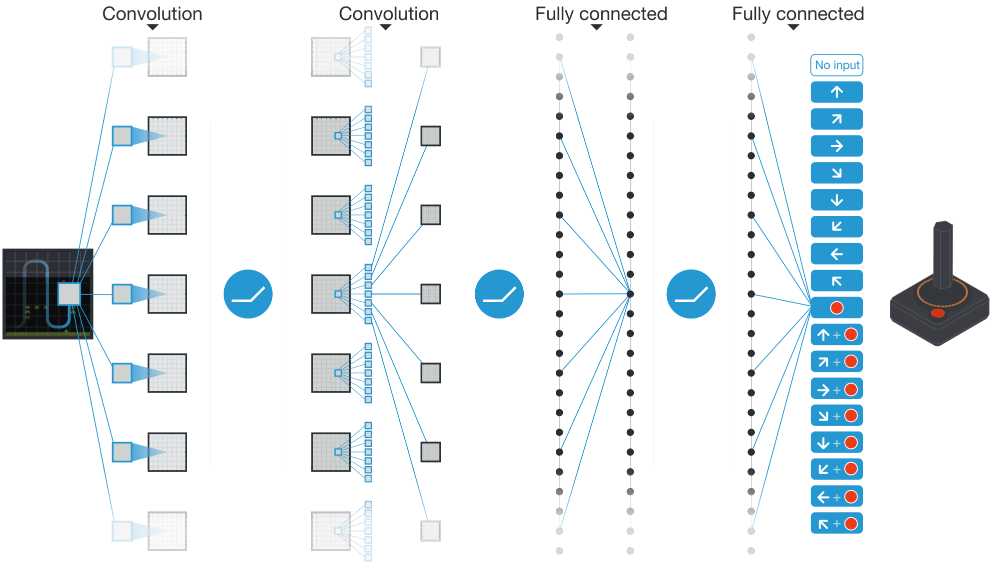
1. A batch of training data is given to both networks.
   - The Q network takes the current state and action from each data sample and predicts the Q value for that particular action.
   - The Target network takes the next state from each data sample and predicts the best Q value out of all actions that can be taken from that state.
2. The loss function at iteration i is defined as

$$J_i(\theta_i) = \hat{\mathbb{E}}_{(s,a,r,s')\sim U(S)}\left[\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]\right]$$

   where $U(S)$ is uniform distribution from the training set S and θ_i^- is the target network parameters.
3. Only Q-network is trained and the target network is fixed.
4. Every C steps, the weights of Q-network is copied to the target network.

Deep Q-Network (Mnih et al. 2015)

Q-Network architecture

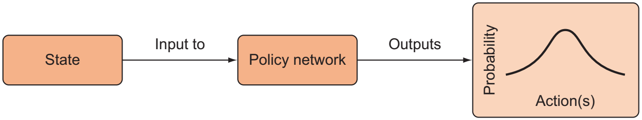


Policy-Based Deep RL

Deep Policy Networks

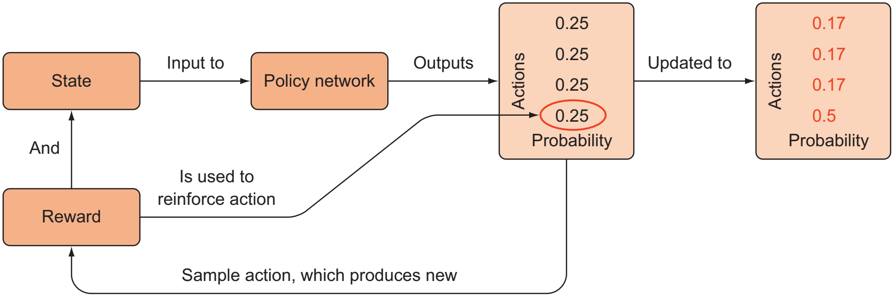
1. Represent policy by deep network with weights w : $a = \pi(a|s, w)$

Policy network



2. Define objective function as total discounted reward: $L(w) = \mathbb{E}\left[\sum_{k=0} \gamma^k r_{k+1} \mid \pi(\Delta, w)\right]$
3. Optimize objective end-to-end by SGD (adjust policy parameters to achieve more reward)

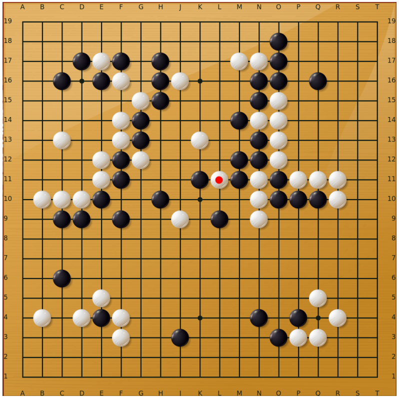
Policy network



AlphaGo

Go

- More than 2500 years old
- Considered the hardest classical board game
- Played on 19×19 board simple rules:
  - Players alternately place a stone
  - Surrounded stones are removed
  - Player with more territory wins



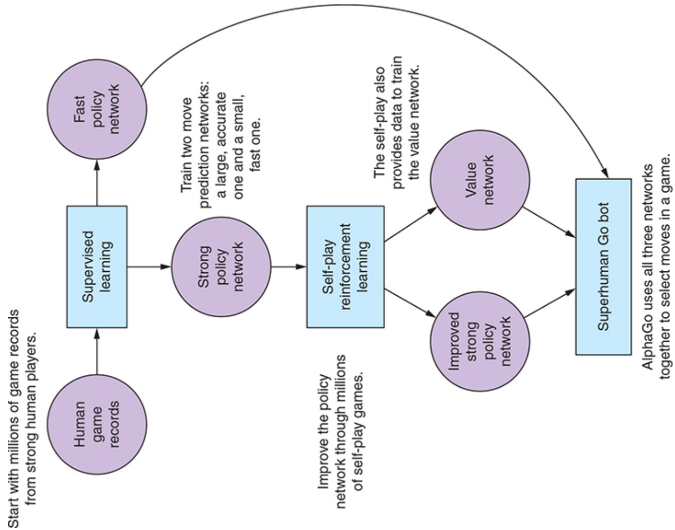
1. Deep learning + Monte Carlo Tree Search(MCTS) + High Performance Computing (Silver, Huang, et al. 2016; Silver, Schrittwieser, et al. 2017).
2. Learn from 30 million human expert moves and 128,000+ self play games.
3. AlphaGo uses CNNs in which
   - Use policy network to explore better (and fewer) moves.
   - Use value network to estimate lower branches of tree in MCTS.

Go board states

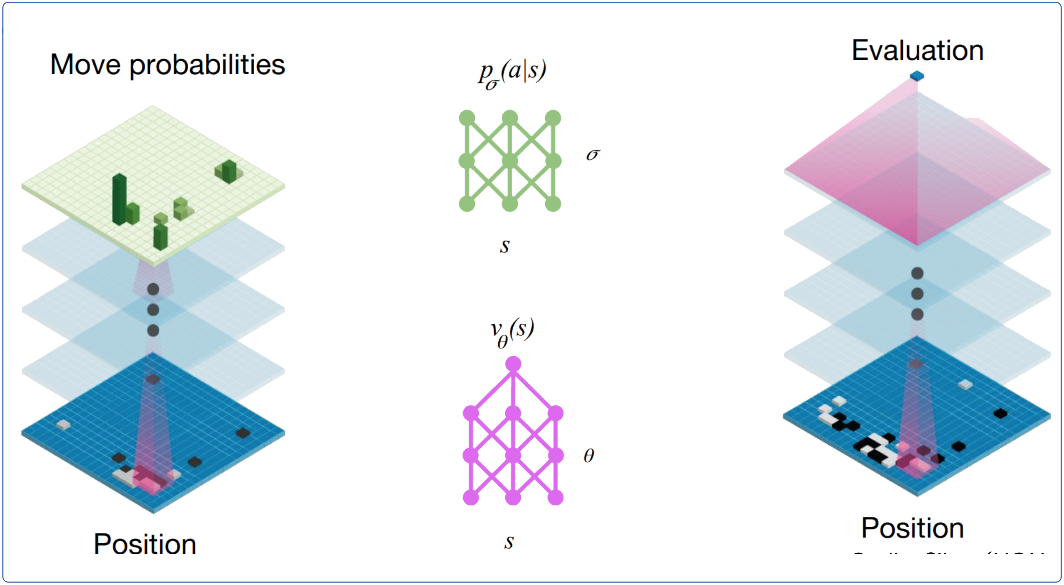
Extended Data Table 2 | Input features for neural networks

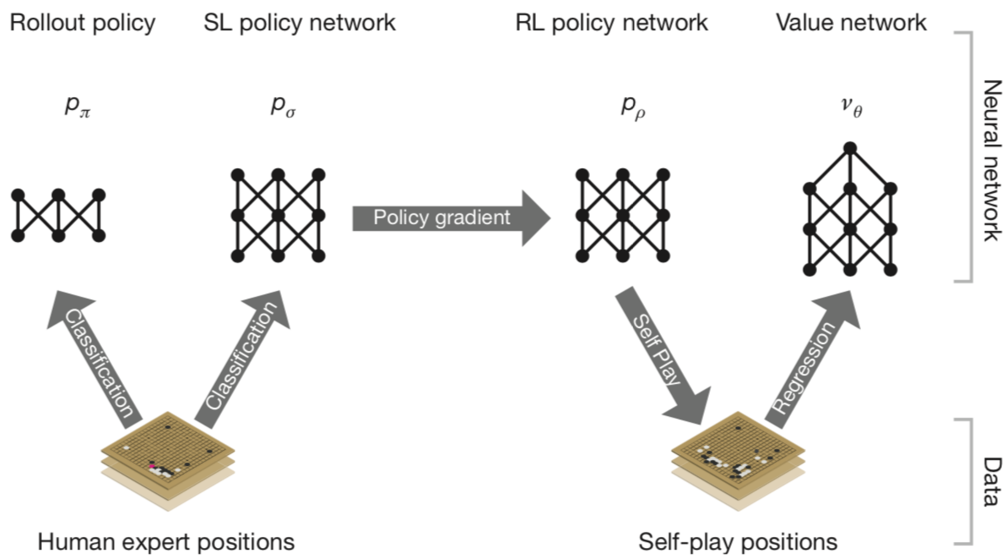
Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

AlphaGo flow diagram



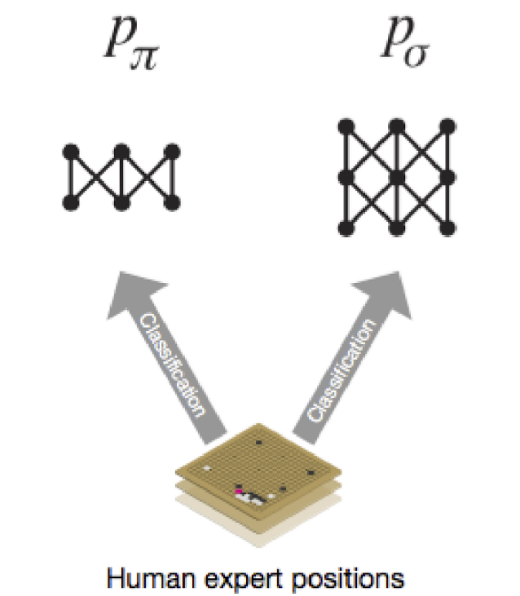
Separate 12-layer CNNs with ReLU activations




Rollout policy
SL policy network
RL policy network
Value network
p_π
p_σ
p_ρ
v_θ
Policy gradient
Neural network
Classification
Classification
Self Play
Regression
Data
Human expert positions
Self-play positions

Training AlphaGo networks (step 1)

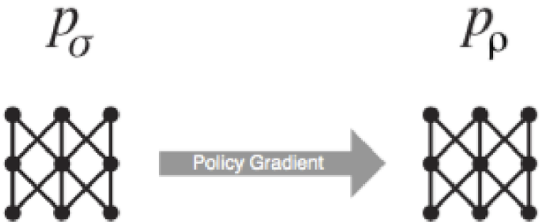
1. Learn to predict human moves
2. Used a large database of online expert games.
3. Learned two versions of the neural network:
   - A fast network P_π for use in evaluation
   - An accurate network P_σ for use in selection.



Training AlphaGo networks (step 2)

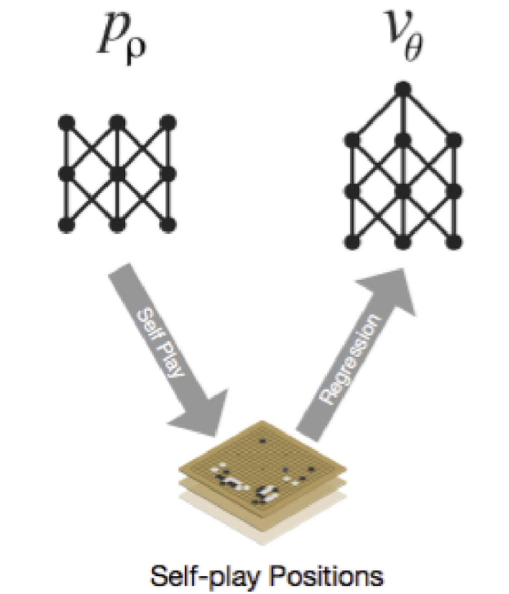
Improve P_σ (accurate network)

1. Run large numbers of self-play games.
2. Update P_ρ using reinforcement learning.
3. Weights updated by stochastic gradient descent.



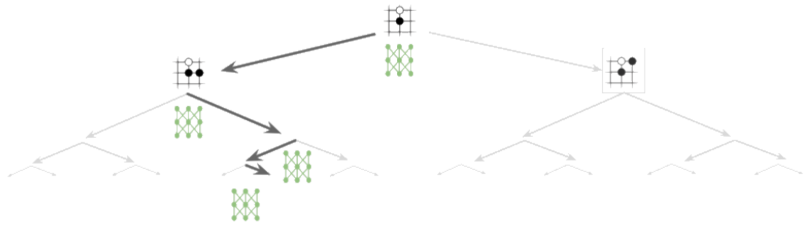
Training AlphaGo networks (step 3)

1. Learn a better board evaluation V_θ
2. use random samples from the self-play database
3. prediction target: probability that black wins from a given board



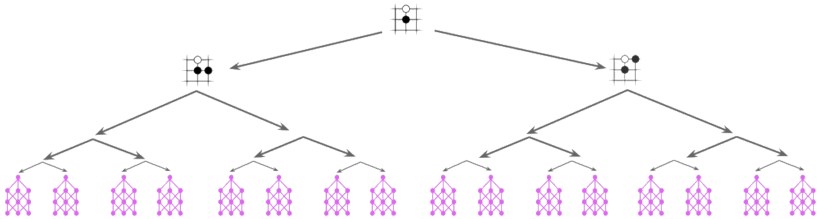
Policy Network and MCTS Search Breadth

1. Approximate leaf values in MCTS using rollouts specified by policy network instead of MC random rollouts
2. Reduce the search breadth in MCTS



Value Network and MCTS Search Depth

1. Approximate leaf values in MCTS using a value network instead of MC rollouts
2. Reduce the search depth in MCTS



Reading

Readings

1. Chapters 1 to 6 and 13 of Reinforcement Learning: An Introduction[1].
2. Paper An Introduction to Deep Reinforcement Learning[2].

[1] Richard S. Sutton and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. Second edition. The MIT Press.
[2] Vincent Francois-Lavet et al. (2018). "An Introduction to Deep Reinforcement Learning". In: *Foundations and Trends in Machine Learning* 11.3-4, pp. 219–354.

References i

- Francois-Lavet, Vincent et al. (2018). "An Introduction to Deep Reinforcement Learning". In: *Foundations and Trends in Machine Learning* 11.3-4, pp. 219–354.
- Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, pp. 529–533.
- Silver, David, Aja Huang, et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587, pp. 484–489.
- Silver, David, Julian Schrittwieser, et al. (2017). "Mastering the game of Go without human knowledge". In: *Nature* 550.7676, pp. 354–359.
- Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement Learning: An Introduction*. Second edition. The MIT Press.

Questions?