CSE 574 Planning and Learning Methods in AI

Ransalu Senanayake

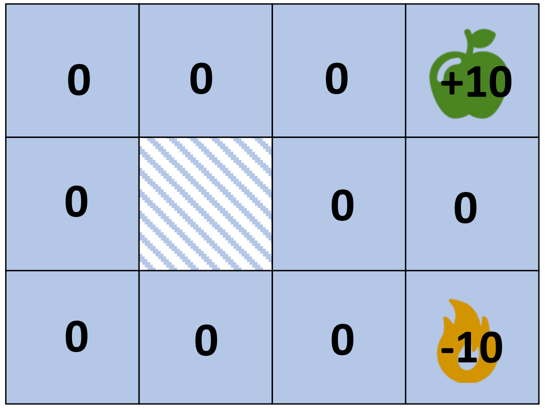
Week 5

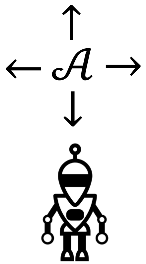
Extracting a Policy

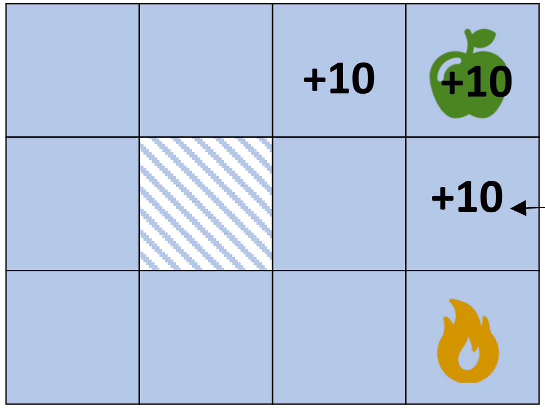
- Brute force
  - For every possible policy, compute the reward – not computationally feasible
- Dynamic programming (DP)
  - DP: Breaking down a superproblem into small subproblems and solving iteratively. The optimality in subproblems guarantees the optimality in the superproblem.
  - If we know the *model* (MDP/transition dynamics and rewards), then we can use exact DP solve using *value iteration* or *policy iteration.* It’s not RL.
  - RL is sometimes called approximate dynamic programming. In RL, we either don’t assume a model (*model-free RL*) or learn the model through interactions *(model-based RL).*

Dynamic Programming for MDPs

Rewards



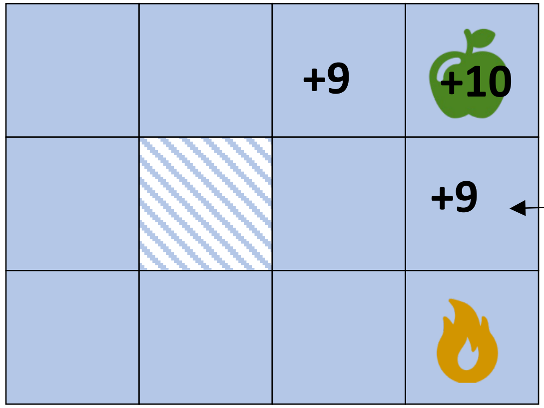




[← = 0,
↑ = +10,
→ = 0,
↓ = −10]

$$V(s_t) = \max_{a \in \mathcal{A}} V(s_{t'})$$

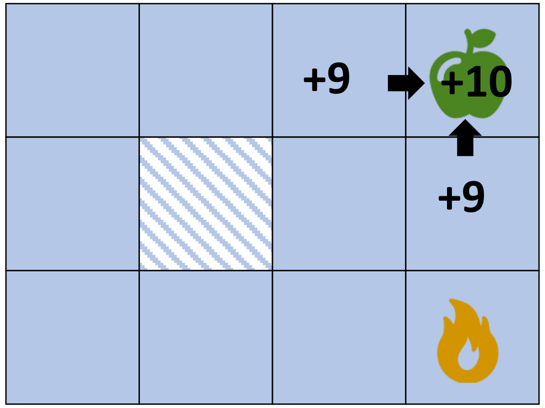
-1 reward for step and 0.9 discount



[← = −1,
↑ = +9,
→ = −1,
↓ = −11]

$$V(s_t) = \max_{a \in \mathcal{A}} (R(s_t, a) + \gamma V(s_{t'}))$$

Dynamic Programming for MDPs



Bellman Equation

$$V(s_t) = \max_{a \in \mathcal{A}}(R(s_t, a) + \gamma V(s_{t'}))$$

$$V^{\pi}(s_t) = \sum_{a \in \mathcal{A}} \pi(s_t|a) Q^{\pi}(s_t, a)$$

Expected return in state, following policy

$$Q^{\pi}(s_t, a) = \sum_{s_{t+1 \in \mathcal{S}}} p(s_{t'}|s_t, a)(r(s_t, a) + \gamma V_{k-1}(s_{t'}))$$

Expected return of following action in state, following policy

Value Iteration

$V(s_0) \leftarrow$ `initialize (e.g., =0), for all` $s \in \mathcal{S}$

for $k = [1:\infty]$

//Improve the value

for each state s

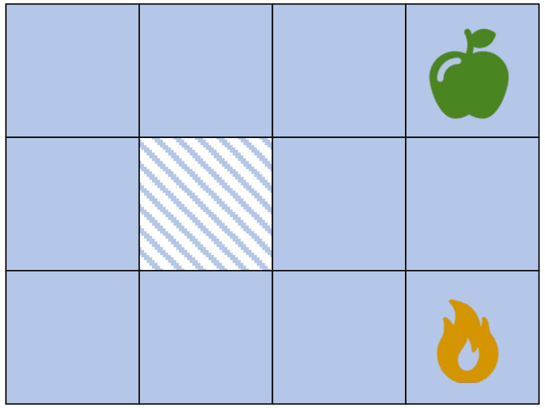
$$V_k(s_t) = \max_{a \in \mathcal{A}} \sum_{s_{t'} \in \mathcal{S}} p(s_{t'}|s_t, a)(r(s_t, a) + \gamma V_{k-1}(s_{t'}))$$

//If no more improvement, extract the policy and return

if $||V_{k-1}(s_t) - V_k(s_t)|| < const.$ *for all* $s \in \mathcal{S}$

$$\pi(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \sum_{s_{t'} \in \mathcal{S}} p(s_{t'}|s_t, a)(r(s_t, a) + \gamma V_{k-1}(s_{t'}))$$

return $[\pi(s)]$ *for all* $s \in \mathcal{S}$



Policy Iteration

```
π(s_0) ← initialize (e.g., =0), for all s ∈ S
for k = [1:∞]
    // Policy evaluation
    // Given the policy, compute value
    for each state s
```

$$V_{k-1}^{\pi_{k-1}}(s_t) = \sum_{s_{t'} \in \mathcal{S}} p(s_{t'}|s_t, \pi_{k-1}(s_t))(r(s_t, \pi_{k-1}(s_t)) + \gamma V_{k-1}^{\pi_{k-1}}(s_{t'}))$$

```
    // Policy improvement
    // Using the newly computed value, compute a new policy
    for each state s
```

$$\pi_k(s_t) = \operatorname*{argmax}_{a \in \mathcal{A}} \sum_{s_{t'} \in \mathcal{S}} p(s_{t'}|s_t, a)(r(s_t, a) + \gamma V_{k-1}^{\pi_{k-1}}(s_{t'}))$$

if $\left|\left|\pi_{k-1}(s_t) - \pi_k(s_t)\right|\right| < const.$ *for all* $s \in \mathcal{S}$

return $[\pi_k(s)]$ *for all* $s \in \mathcal{S}$

Policy iteration demo: http://www.cs.toronto.edu/~lcharlin/courses/60629/reinforcejs/gridworld_dp.html

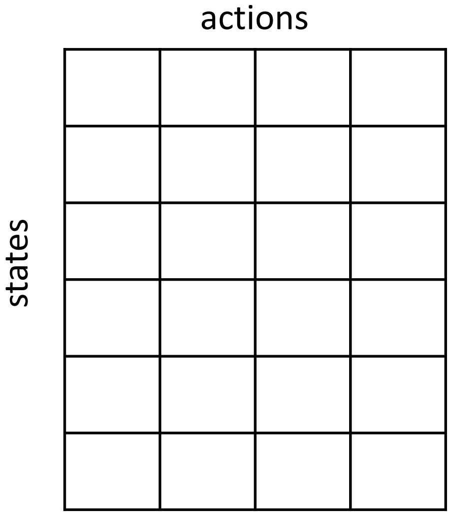
Thoughts

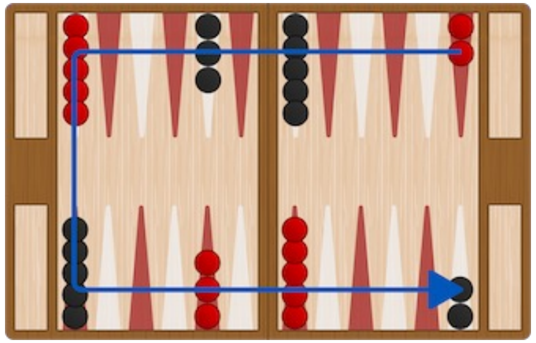
Guarantees convergence

Value iteration can be slow if the state space is large

- Backgammon: 10^20 states
- Chess: 10^40 states
- Go: 10^70 states
- Robotics: continuous state/action spaces

Q table (state-action table)

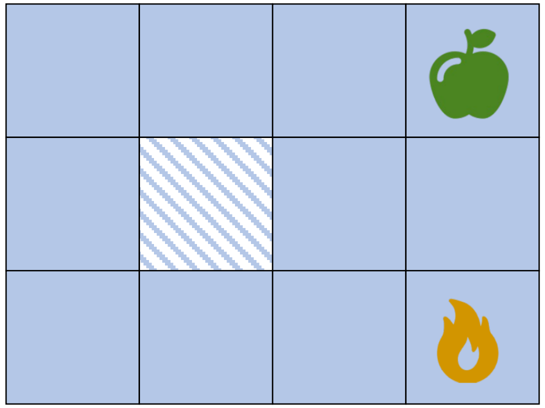


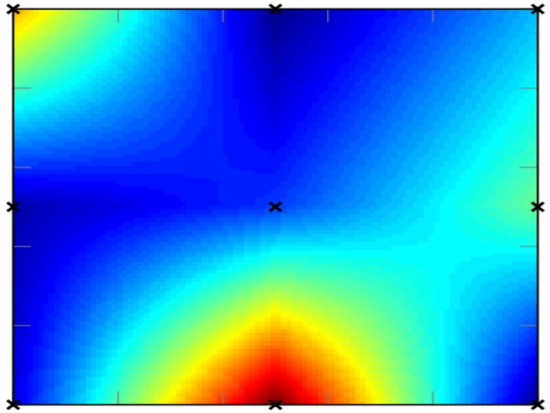


How can we find the policy if the model is unknown? RL.

Approximating the value/policy

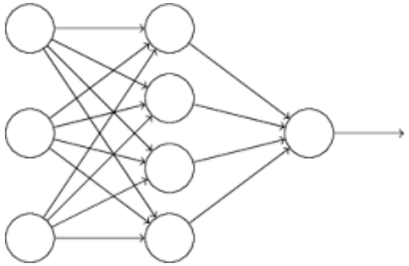
K-NN
Interpolation
Regression
(linear with basis functions, DNN)





Q network

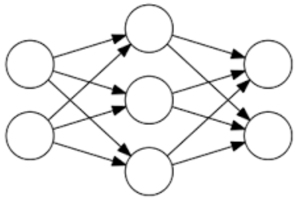
X=Grid locations



y=Q values

Policy network

X=Grid locations



y=policy[
p(up), p(left), p(down),p(right)
]

Pick a state
Predict the its and its neighbors' values from the NN
What should be its value computed from neighbors according to the Bellman eq. (i.e., proxy ground truth)?
Backpropagate the error=MSE(NN prediction-Bellman computation)

Pick a trajectory
Compute the values for each s
Gradient desc p such that higher values are proportional to the desired directions

Deep Q-Network (DQN)

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄ Subscribe

nature > letters > article

Published: 25 February 2015

Human-level control through deep reinforcement learning

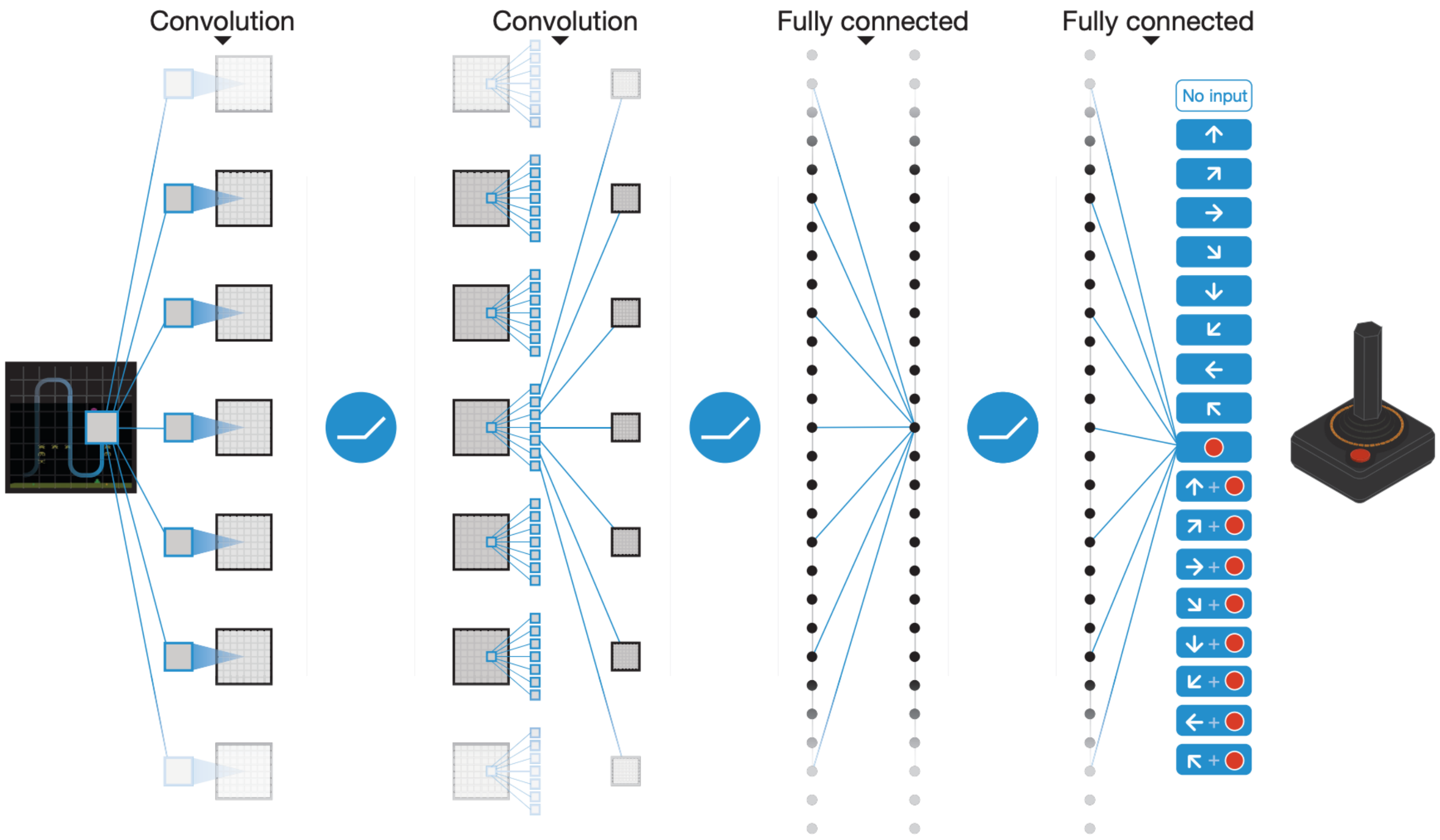
Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

Nature **518**, 529–533 (2015) | Cite this article

Q-learning

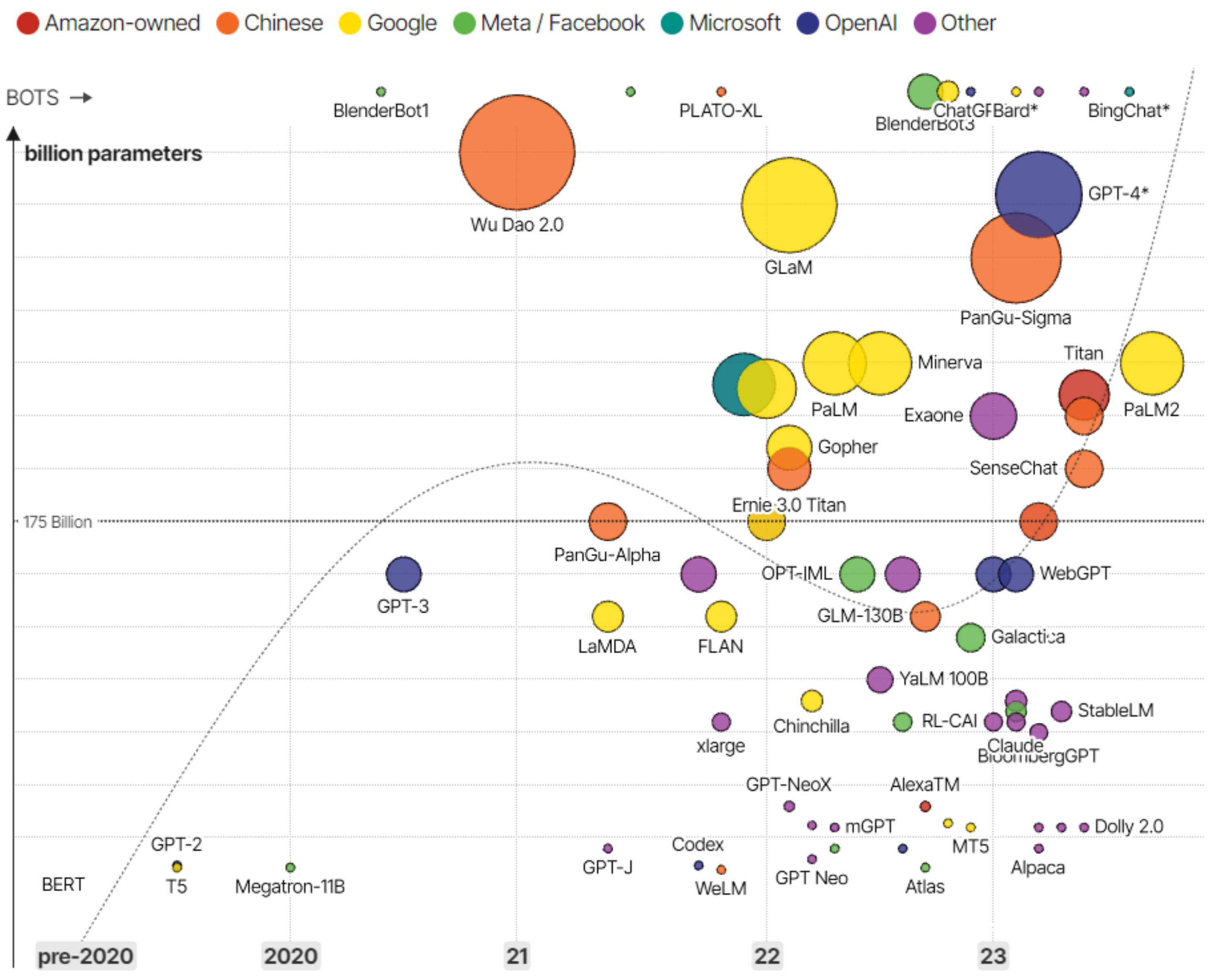
Estimated optimal Old

$$Q'^{(s_t,a_t)} = Q_k(s_t,a_t) + \alpha\underbrace{(r(s_t,a_t) + \gamma.\max(Q(s_{t+1},a_t) - Q(s_t,a_t)))}_{\text{Temporal difference (TD)}}$$


Convolution
Convolution
Fully connected
Fully connected
No input

Deep Q-Network (DQN)

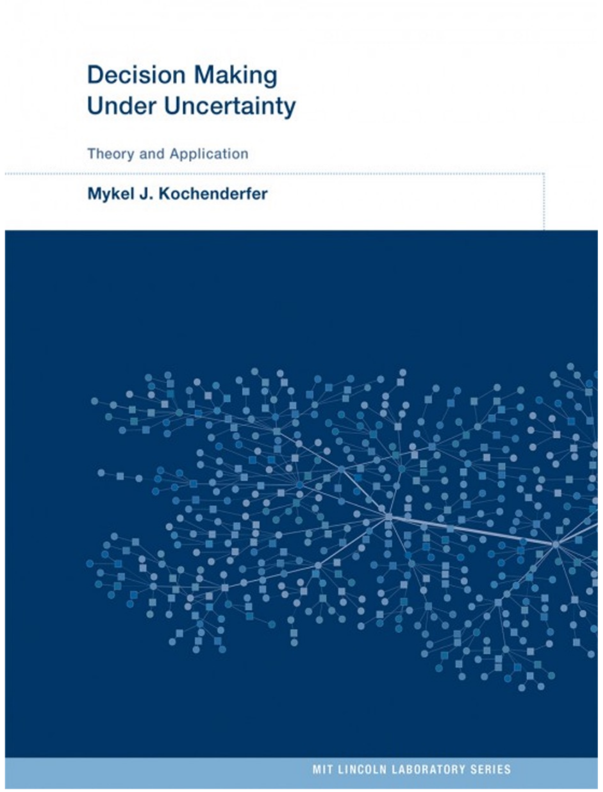
1. Query the current state. NN -> Q value
2. Select an action based on the Q value (use ε-greedy)
3. Collect s’, r
4. Save <s,a,s’,r> in an **experience buffer**
5. Replay (ff the Q network) using several samples from the buffer
6. After several iterations clone the **Q network** in the **target network**

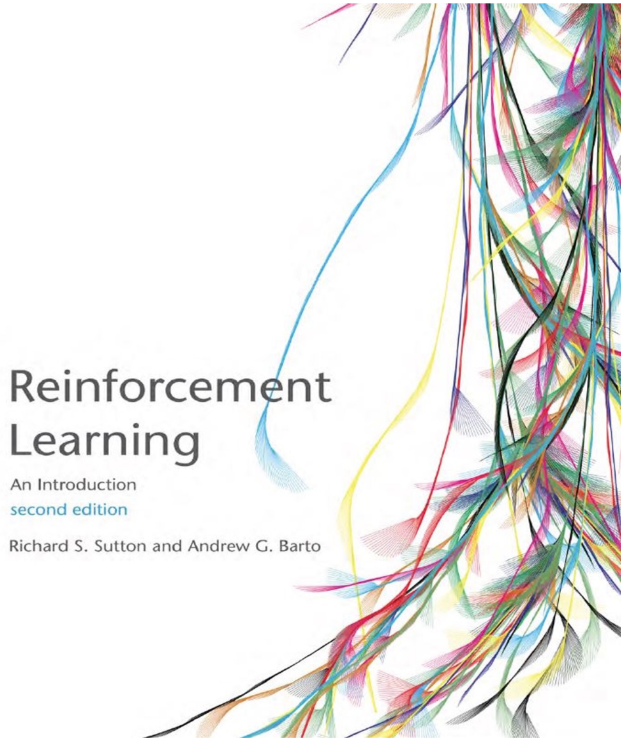


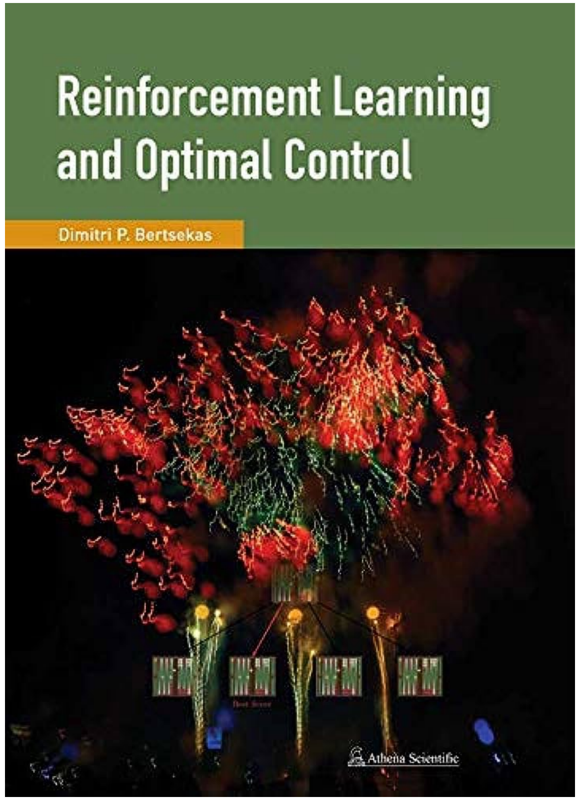
David McCandless, Tom Evans, Paul Barton
Information is Beautiful // May 2023

source: news reports, LifeArchitect.ai
* = parameters undisclosed // see the data

MADE WITH VIZsweet


Decision Making
Under Uncertainty
Theory and Application
Mykel J. Kochenderfer
MIT LINCOLN LABORATORY SERIES


Reinforcement
Learning
An Introduction
second edition
Richard S. Sutton and Andrew G. Barto


Reinforcement Learning
and Optimal Control
Dimitri P. Bertsekas
Athena Scientific