

Reinforcement Learning in Digital Finance

Markov Decision Processes and basics of Temporal
Difference Learning



Funded by
the European Union

Value function approximation [1/2]

- So far, we stayed close to Dynamic Programming paradigm:
 - Replace true value functions V with approximation \bar{V}
 - Several ways to find suitable \bar{V} (e.g., Q-table, features)
 - Finding optimal value functions equates finding optimal policy
- Disadvantages of VFA:
 - Falls apart for continuous- and large action spaces
 - Must evaluate $\bar{V}(s, a)$ for every action a in state s .
 - Indirect and unnatural way of decision-making
 - Dynamic Programming not intuitive for everyone



Lecture topics

- MDP building blocks
- State-, outcome-, and action spaces
- Post-decision states and look-up tables
- Temporal difference learning: Monte Carlo learning, Q-learning and SARSA





Examples of MDPs

State [1/2]

- # containers per type:
[#red, #blue, #green]
- Stock trading:
[cash amount, stock price, amount invested]
- Warehouse reordering:
[# product type 1, # product type 2]
- Often combinatorial: $[y]_{a,b,c \in A \times B \times C}$



State [2/2]

- Include all information needed for...
 - Making decisions
 - Computing rewards
 - Computing state transitions
- State can be both physical state properties and information



Action [1/2]

- Container shipping:
[#red shipped, #blue shipped, #green shipped]
- Stock trading:
[amount stock bought/sold]
- Warehouse reordering:
[# order product type 1, # order product type 2]
- Typically based on (part of) state vector



Action [2/2]

- Don't forget constraints and domains!

Constraint examples

$$a_{red} \leq s_{red}$$

$$a_{stock} \leq c_t$$

$$p_1 + p_2 + a_1 + a_2 \leq C$$

- ▶ Don't ship more red containers than available
- ▶ Can't buy more stock than current cash allows
- ▶ Don't exceed warehouse capacity

Domain examples

$$a_{red} \in [0, s_{red}]$$

$$a_{stock} \in [-s_{stock}, c_t]$$

$$a_1, a_2 \in \mathbb{N}$$

- ▶ Ship no more than existing containers
- ▶ Buy/sell according to cash position
- ▶ Only order positive integer amounts of product



Reward

- $r_t = \begin{cases} -800 & \text{if } x = \{1,0,0\} \\ -700 & \text{if } x = \{0,1,0\} \\ \dots \\ -1200 & \text{if } x = \{1,1,0\} \\ -1900 & \text{if } x = \{1,1,1\} \end{cases}$
- $r_t = \text{amount stock sold} - \text{amount stock bought}$
- $r_t = \sum_{i \in \{1,2\}} p_i \omega_i - \sum_{i \in \{1,2\}} c_i a_i$
- Capture *direct* rewards of action
 - Different from value functions and objective function!



Probability function [1/2]

- Transition matrix (from s_t to s_{t+1} given action a_t):

$$\begin{bmatrix} 0.2 & 0.8 \\ 0.9 & 0.1 \end{bmatrix}$$

- Transition function. Separate outcome ω from action a
 - Can get very complex
 - Often needed to fill transition matrix

$$p_{\omega}^{\Omega} = \beta \cdot p_f^F p_g^G \cdot \prod_{d \in \mathcal{D}, r \in \mathcal{R}, k \in \mathcal{K}} \left(\left(p_d^{D^F} p_r^{R^F} p_k^{K^F} \right)^{\tilde{F}_{d,r,k}^{\omega}} \left(p_d^{D^G} p_r^{R^G} p_k^{K^G} \right)^{\tilde{G}_{d,r,k}^{\omega}} \right)$$

where

$$f = \sum_{d \in \mathcal{D}, r \in \mathcal{R}, k \in \mathcal{K}} \tilde{F}_{d,r,k}^{\omega}$$

$$g = \sum_{d \in \mathcal{D}, r \in \mathcal{R}, k \in \mathcal{K}} \tilde{G}_{d,r,k}^{\omega}$$

$$\beta = \frac{f!}{\prod_{d \in \mathcal{D}, r \in \mathcal{R}, k \in \mathcal{K}} \left(\tilde{F}_{d,r,k}^{\omega}! \right)} \cdot \frac{g!}{\prod_{d \in \mathcal{D}, r \in \mathcal{R}, k \in \mathcal{K}} \left(\tilde{G}_{d,r,k}^{\omega}! \right)}$$



Probability function [2/2]

- Probability function part of MDP, but not always necessary for RL
- Explicit probability function does not always exist
 - Games: generate observations through interaction with environment, no need to understand dynamics of game
 - Financial markets: may observe stock price changes directly from the market
- Reinforcement learning is very data-hungry
 - A data generator may be needed, e.g., generating synthetic stock data rather than relying on a single historical time series

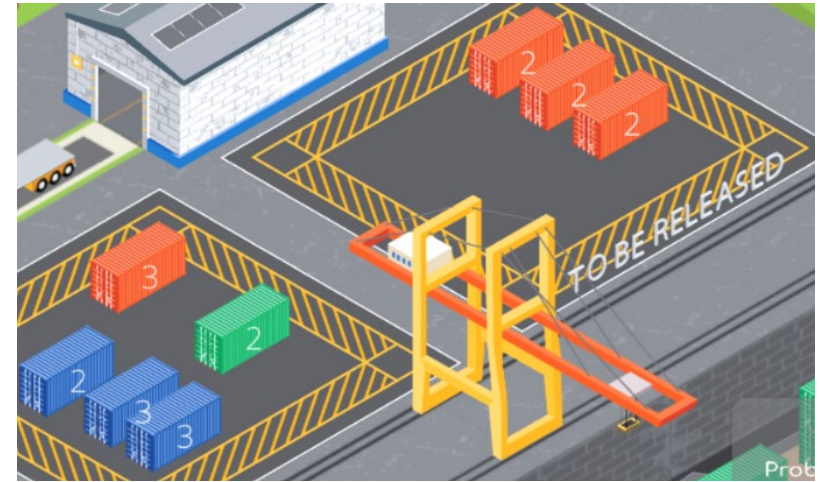




The curses of dimensionality

Example – Trucks and barges

- We manage transport from a container yard
- Every day, containers may be shipped by barge, train or truck.
- Barge and train have scale advantages
- Truck always a viable back-up option
- Container properties:
 - Due date τ
 - Destination d
 - Pre-announced or ready β
- Containers arrive part stochastically, part pre-announced
- How to allocate containers to transport modes as cheaply as possible?



Example – Computational complexity

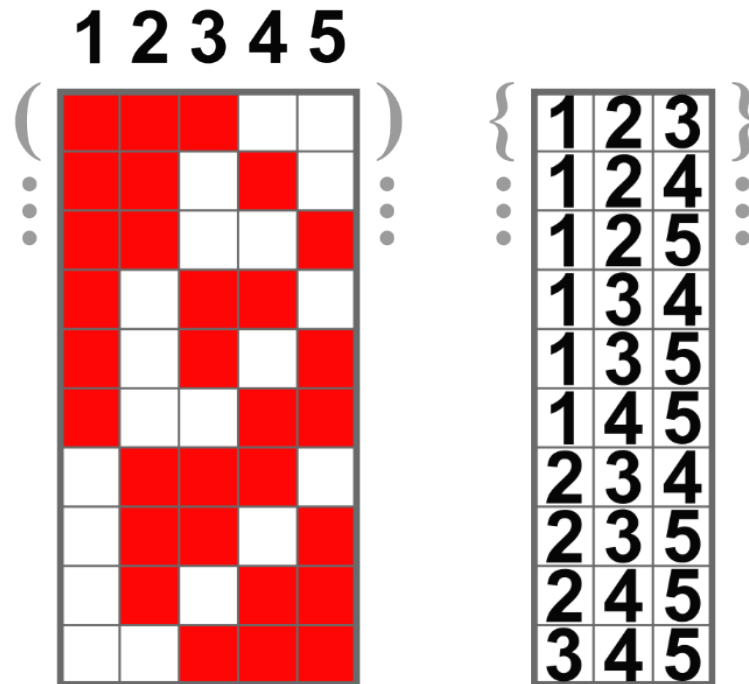
- Container type: Unique destination (3), due date (5), present/pre-announced (2)
- State: number of containers per type currently present
 - Vector with $3*5*2=30$ dimensions
 - E.g., holding 12 containers of different types yields $\frac{30!}{12!(30-12)!} = 86,493,225$ possible states
- Action per container: hold, truck, train, barge
 - 4^n actions (e.g., $4^{12} > 16$ million actions)
- Outcome: every possible permutation of containers
 - Again, extremely large number of outcomes



Example – Combinations

- Combinations with 5 containers, 3 container types

- $[5, 0, 0]$
- $[4, 1, 0]$
- $[3, 2, 0]$
- $[3, 1, 1]$
- $[2, 2, 1]$



$$\binom{n}{k} = \frac{n(n-1) \cdots (n-k+1)}{k(k-1) \cdots 1}$$

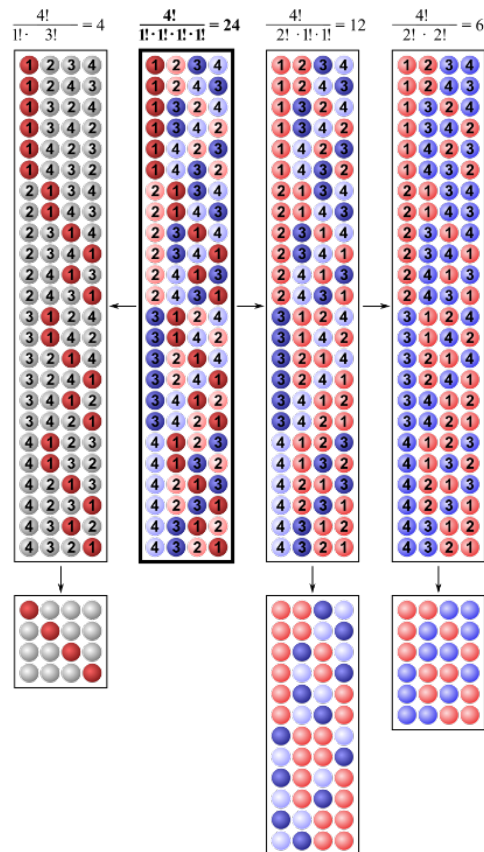
<https://en.wikipedia.org/wiki/Combination>



Example – Permutations

- Many permutations per combination

- $[3, 2, 0]$
- $[3, 0, 2]$
- $[0, 3, 2]$
- $[0, 2, 3]$
- $[2, 0, 3]$
- $[2, 3, 0]$



$$\binom{n}{m_1, m_2, \dots, m_l} = \frac{n!}{m_1! m_2! \dots m_l!} = \frac{\left(\sum_{i=1}^l m_i\right)!}{\prod_{i=1}^l m_i!}$$



Exact sizes require statistics

- Draws from jar of marbles
 - Order of draw does not matter (combination)
 - Order of draw does matter (permutation)
 - Draw with/without replacement
 - Dealing with duplicates?
- Domain of **combinatorics** → can get quite advanced!
- For continuous state spaces (e.g., stock prices), outcomes are of course infinite



Resolving the curses of dimensionality

- RL offers solutions for each of the three curses
- Some mathematical guarantees exist...
 - However, only in the limit, infinite sampling, etc.
 - Realistically, RL algorithms are suboptimal
- Every simplification has an impact, but inevitable to find solutions
 - Tradeoff between computational effort and quality



I: Outcome space Ω

- Dynamic Programming requires evaluating *all* outcomes $\omega_t \in \Omega_t$ for every state-action pair (s_t, a_t)
- **How to resolve?**



I: Outcome space Ω



- Randomly sample from potential outcomes
 - Generate random arrivals of containers $\omega_t \in \Omega_t$
 - Again some vector, e.g., $\omega_t = [2, 3, 1]$
- Unbiased estimate of future costs
 - Theoretically, should converge to the true value
 - Law of large numbers \rightarrow infinite sample approximates mean
- Learn values for state-action pairs
- We transform the stochastic equation into a deterministic one:
 - Estimate $\bar{V}(s_t, a_t)$ instead of $\sum_{\omega_t \in \Omega_t} V_{t+1}(s_{t+1} | s_t, a_t, \omega_t)$



II: State space \mathcal{S}

- Dynamic Programming requires (repeatedly) visiting every state $s \in \mathcal{S}$ to learn its value function $V(s)$.
- **How to resolve?**



II: State space \mathcal{S}

- We can't observe every state to learn its value
- Many unique states, but also similarities:
 - For example: many blue containers comparable to many red or green containers
 - State $[3,4,2]$ has properties similar to $[3,3,2]$
 - We can derive generic insights from observing states

- **Feature design** \rightarrow Explanatory variables $\phi_f: (s_t, a_t) \mapsto \mathbb{R}$ that capture essence of state's value

$$\tilde{V}(s_t, a_t) = \sum_{f \in \mathcal{F}} \theta_f(s_t, a_t) \cdot \phi_f$$

- Update weights θ_f after each observation (regression procedure)



III: Action space \mathcal{A}

- Dynamic programming requires evaluating every possible action $a \in \mathcal{A}(s)$ to be taken in a given state s .
- **How to resolve?**



III: Action space \mathcal{A}

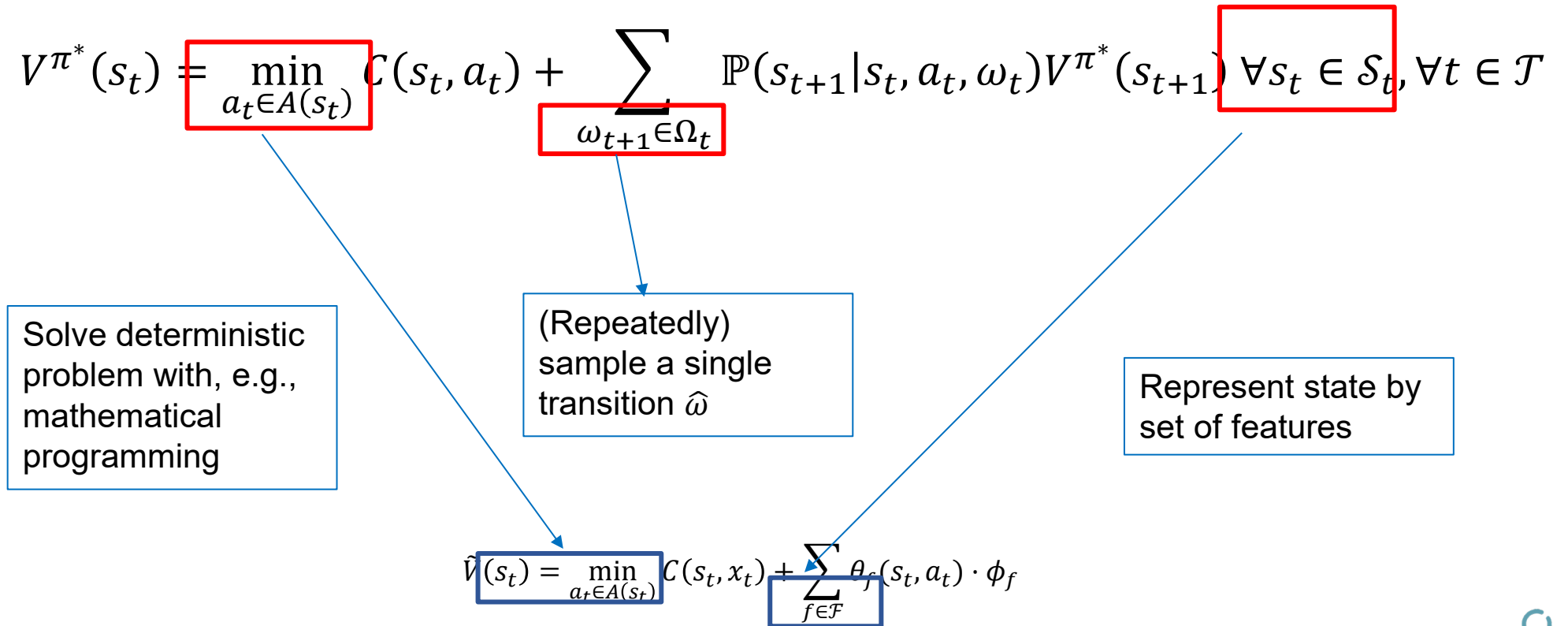
- Stochastic equation has been transformed into deterministic one (recall outcome space)
 - Only one estimated downstream value per action
 - Perhaps small enough to enumerate now
- Alternative: mathematical programming
 - Many OR problems can be described as linear program
 - Preserves optimality property

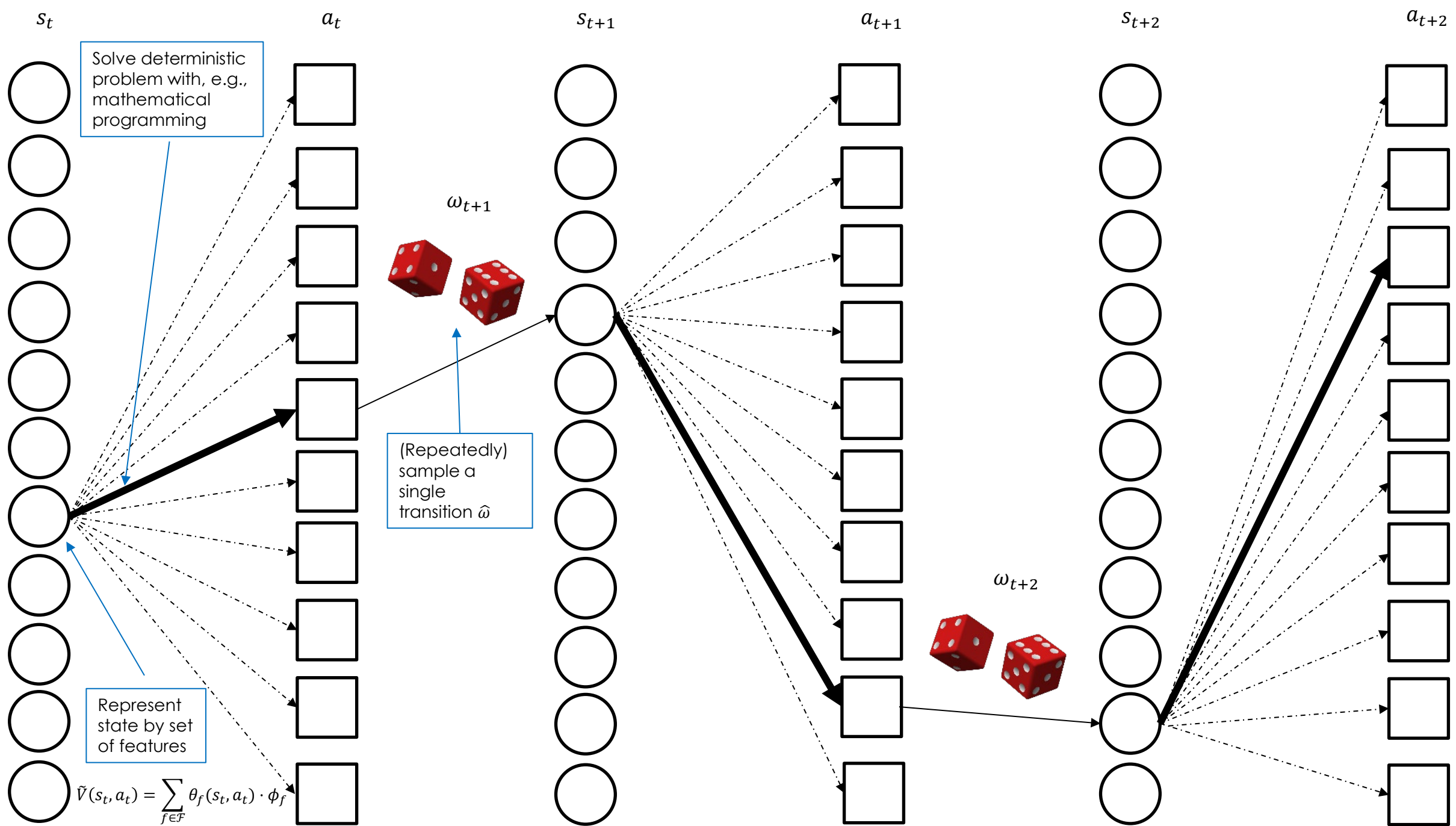
$$\tilde{V}(s_t) = \min_{a_t \in \mathcal{A}(s_t)} C(s_t, a_t) + \sum_{\omega_{t+1} \in \Omega_t} \theta_f(s_t, a_t) \cdot \phi_f$$

- If all else fails, use heuristics and assumptions



Curses of dimensionality – To summarize





The background is a dark teal color with a complex, glowing network of light blue lines and nodes. The nodes are small squares, and the lines connect them in a web-like pattern, creating a sense of depth and connectivity. The text is centered in the middle of the image.

Outcome space – Post-decision states

Curse I: Outcome space

- For now, let's ignore state space and action space
 - Focus on outcome space
 - Perform action a_t in state s_t
 - Randomly sample outcome $\hat{\omega} \in \Omega$
 - Observe value corresponding to state
- Before continuing, let's provide a deeper perspective on **outcomes**.



Transition function

- During transition from s_t to s_{t+1} , quite a lot happens!
 - We take decision based on available information at time t
 - After the decision, time moves forward (discrete step t to $t + 1$)
 - *During* step, new system information is revealed.
 - We end up in a new state
- In a transition function:

$$s_{t+1} \leftarrow f(s_t, a_t, \omega_t)$$



Transition function – Deterministic

- It can be useful to split the transition into two parts
- Deterministic transition (state-action pair, post-decision state)
 - Based on state and action
 - No time impact

$$s_t^a \leftarrow f^{(1)}(s_t, a_t)$$

- Already lots of information in this part
 - If we ship many containers today, we probably have few tomorrow
 - If we sell Shell today, fluctuation won't affect portfolio tomorrow



Transition function – Post-decision state

- Result s_t^a is known as the **post-decision state**
 - Sort of ‘intermediate state’ between t and $t + 1$
 - Includes effects of action, but not of new information
- Alternative is a state-action pair (s_t, a_t)
- In Value Function Approximation, we aim to learn value for state-action pair or post-decision state
 - **Partial predictor** without needing a time step
 - If we sample infinite outcomes, $\tilde{V}(s_t, a_t)$ should approach true $V(s_t, a_t)$



Transition function – Stochastic

- Stochastic transition
 - Based on post-decision state and random outcome
 - Includes time step

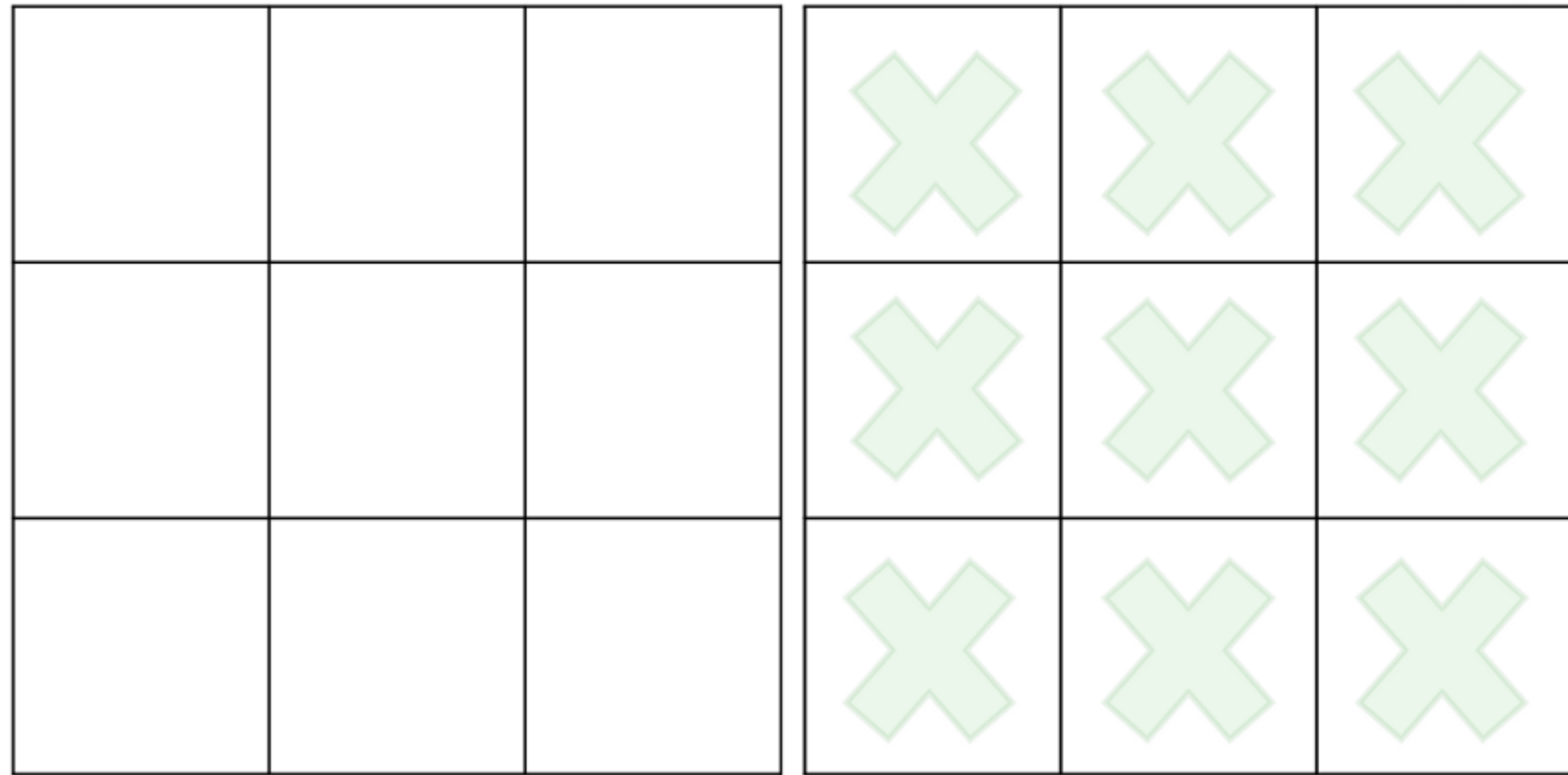
$$s_{t+1} \leftarrow f^{(2)}(s_t^a, \omega_t)$$

- Needed to move forward in time
 - We might know the distribution of the transition, but not the actual realization
 - In RL, we use Monte Carlo simulation to sample outcomes



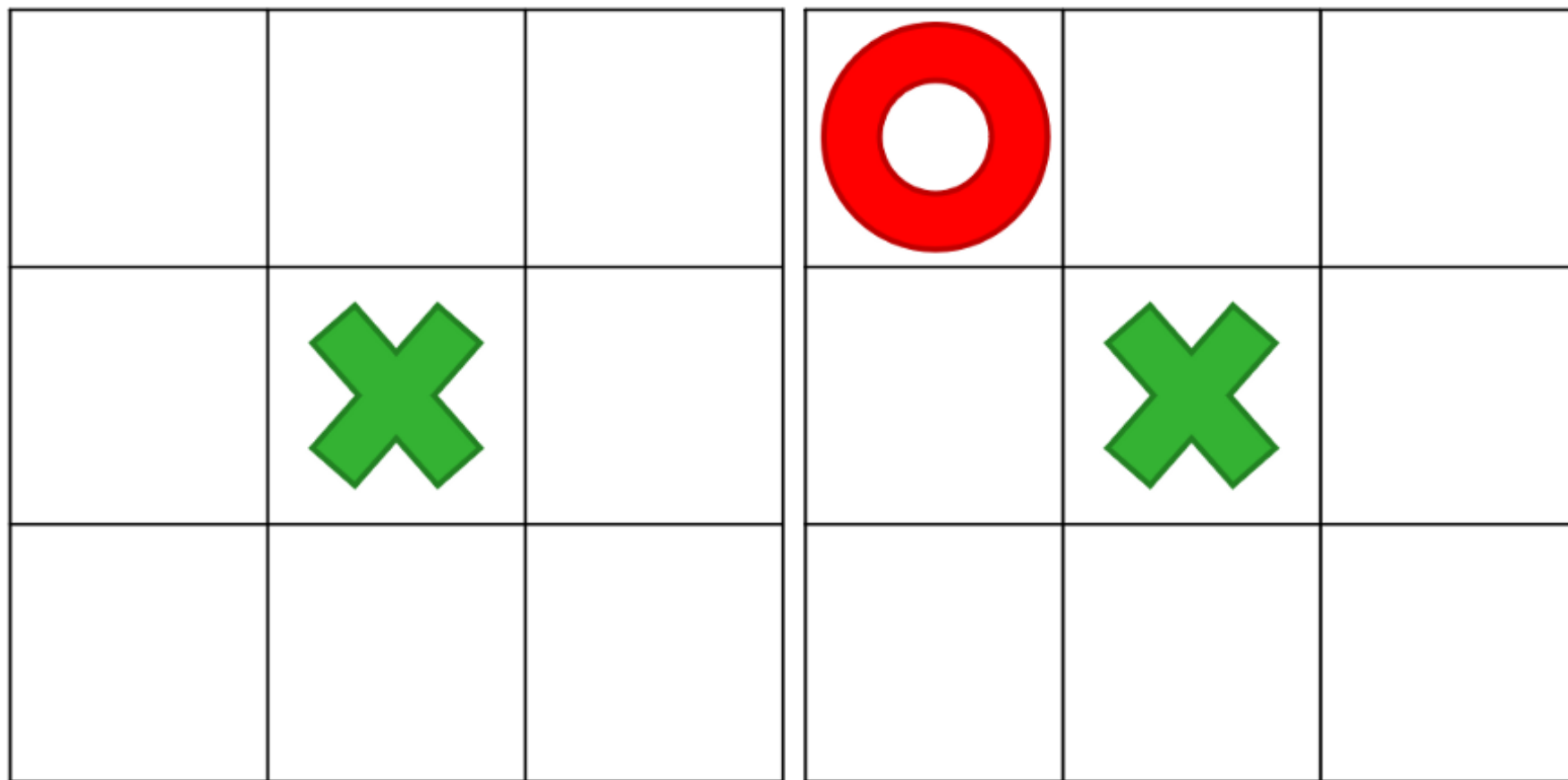
Post-decision state – Example

Initially, all nine moves are possible (deterministic transition $f^{(1)}$)



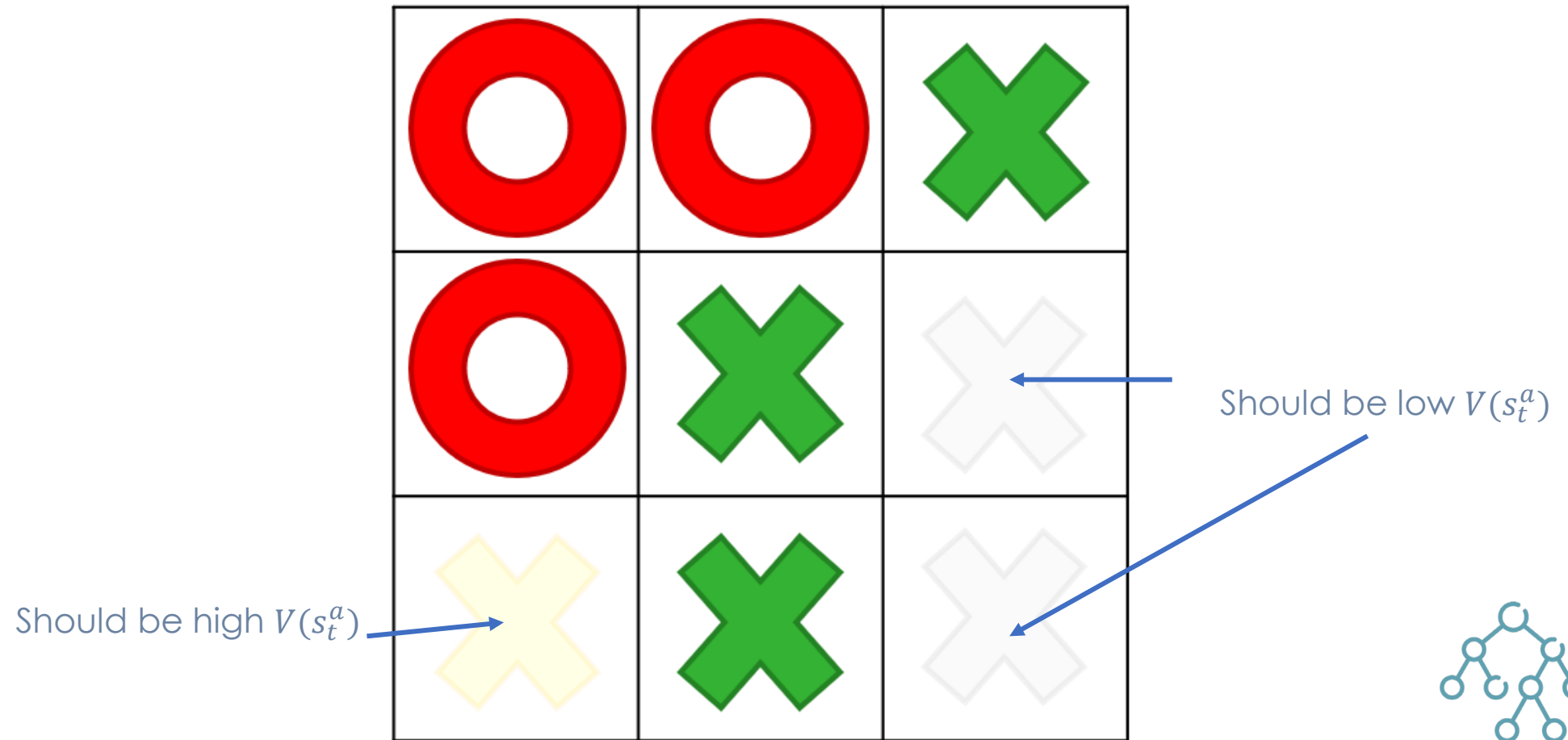
Post-decision state – Example

After picking a move, opponent moves (stochastic transition $f^{(2)}$)



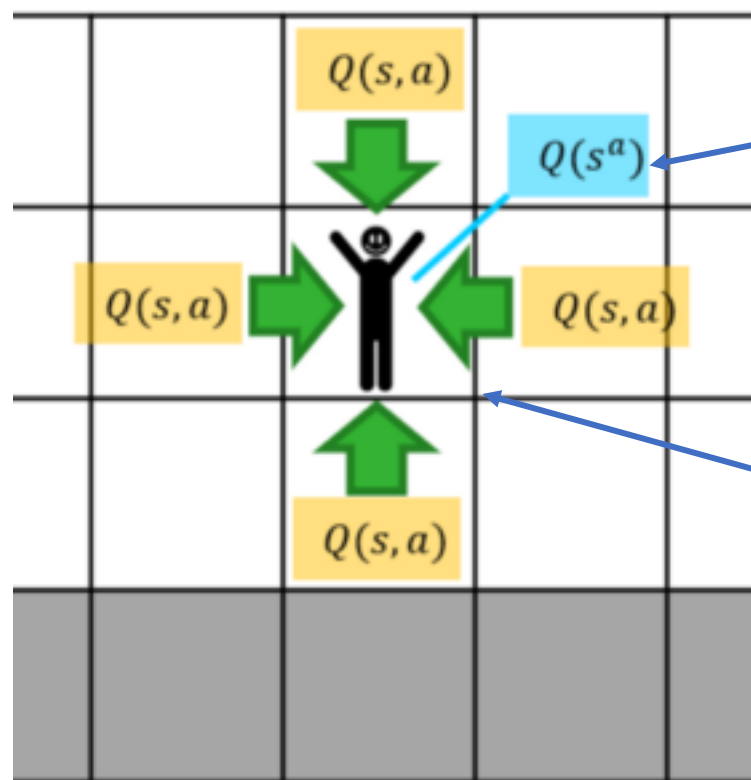
Post-decision state – Example

Near the end, only a few post-decision states are reachable



Post-decision state – Example

- Sometimes post-decision state and state-action pairs are equivalent, but not always



Single value functions for post-decision state. Only the tile the agent lands on matters

Four value functions for state-action pairs. The tile from which the agent moves affects the value.



Post-decision state – Example

- Computation often straightforward (e.g., + order, - demand)

$$\begin{array}{ccc} s_t & a_t & s_t^a \\ \begin{bmatrix} 11 \\ 0 \\ 12 \\ 88 \\ 7 \\ 15 \\ 112 \\ 54 \\ 68 \\ 17 \end{bmatrix} & + \begin{bmatrix} 20 \\ 8 \\ 5 \\ 12 \\ 0 \\ 5 \\ 50 \\ 25 \\ 0 \\ 13 \end{bmatrix} & = \begin{bmatrix} 31 \\ 8 \\ 17 \\ 100 \\ 7 \\ 20 \\ 162 \\ 79 \\ 68 \\ 30 \end{bmatrix} \end{array}$$
$$\begin{array}{ccc} s_t^a & \omega_t & s_{t+1} \\ \begin{bmatrix} 31 \\ 8 \\ 17 \\ 100 \\ 7 \\ 20 \\ 162 \\ 79 \\ 68 \\ 30 \end{bmatrix} & - \begin{bmatrix} 10 \\ 1 \\ 5 \\ 73 \\ 0 \\ 20 \\ 88 \\ 13 \\ 24 \\ 16 \end{bmatrix} & = \begin{bmatrix} 21 \\ 7 \\ 12 \\ 27 \\ 7 \\ 0 \\ 74 \\ 66 \\ 44 \\ 14 \end{bmatrix} \end{array}$$



Q-values

- **Q-value**: downstream value for state-action pair
 - Store value for each state-action pair $\rightarrow Q(s_t, a_t)$
 - For post-decision state $\rightarrow \bar{Q}(s_t^a)$
- Q-values are learned by repeated observation
 - Take action a_t in state s_t , sample outcome ω_t , observe reward
 - With sufficient samples, values should average
 - When you have good Q-values, you can take good decisions



Outcome space 'resolved'

- $Q(s_t, a_t)$ is computationally much simpler than $\sum_{\omega_t \in \Omega_t} V_{t+1}(s_{t+1} | s_t, a_t, \omega_t)$
- Straightforward deterministic optimization problem:

$$\tilde{V}(s_t) = \min_{a_t \in A(s_t)} C(s_t, a_t) + Q(s_t, a_t)$$

- Computationally this matters a lot
 - No loop over Ω_t for each decision
 - However, sampling strategy does not scale infinitely



Monte Carlo Learning

Lookup table

- Explicitly store value per state-action pair.
- Deterministic value rather than evaluating all probabilistic outcomes $\omega \in \Omega$
- Still must enumerate all states and actions.
- Observe state-action pairs multiple times for decent estimate.

	actions			
	a_0	a_1	a_2	\dots
states				
s_0	$Q(s_0, a_0)$	$Q(s_0, a_1)$	$Q(s_0, a_2)$	\dots
s_1	$Q(s_1, a_0)$	$Q(s_1, a_1)$	$Q(s_1, a_2)$	\dots
s_2	$Q(s_2, a_0)$	$Q(s_2, a_1)$	$Q(s_2, a_2)$	\dots
\vdots	\vdots	\vdots	\vdots	\vdots



How to determine Q-value?

- Remember, Q-value is simply expected downstream value
 - Collect many sample trajectories originating from a given state-action pair (s_t, a_t) and store rewards
 - Repeating many times gives unbiased estimate of expected rewards



Monte Carlo Reinforcement Learning

- Sample entire trajectory, then update
 - Compute cumulative rewards G_t

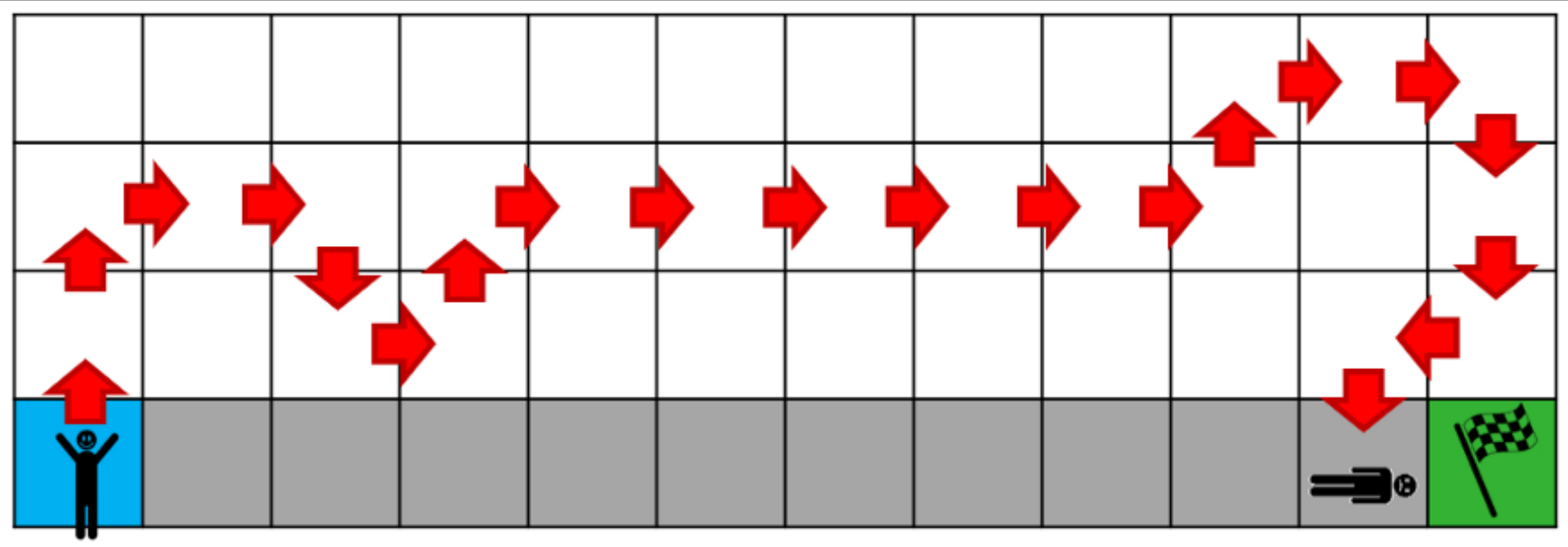
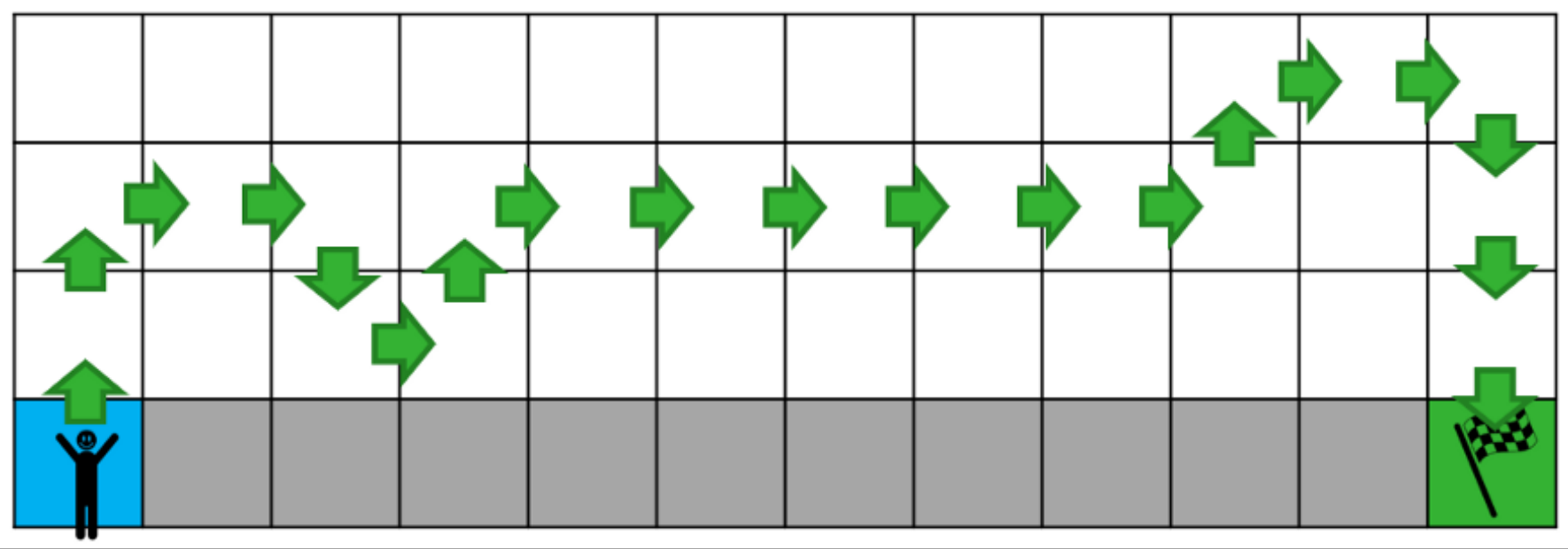
$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots + \gamma^{T-t} R_T$$

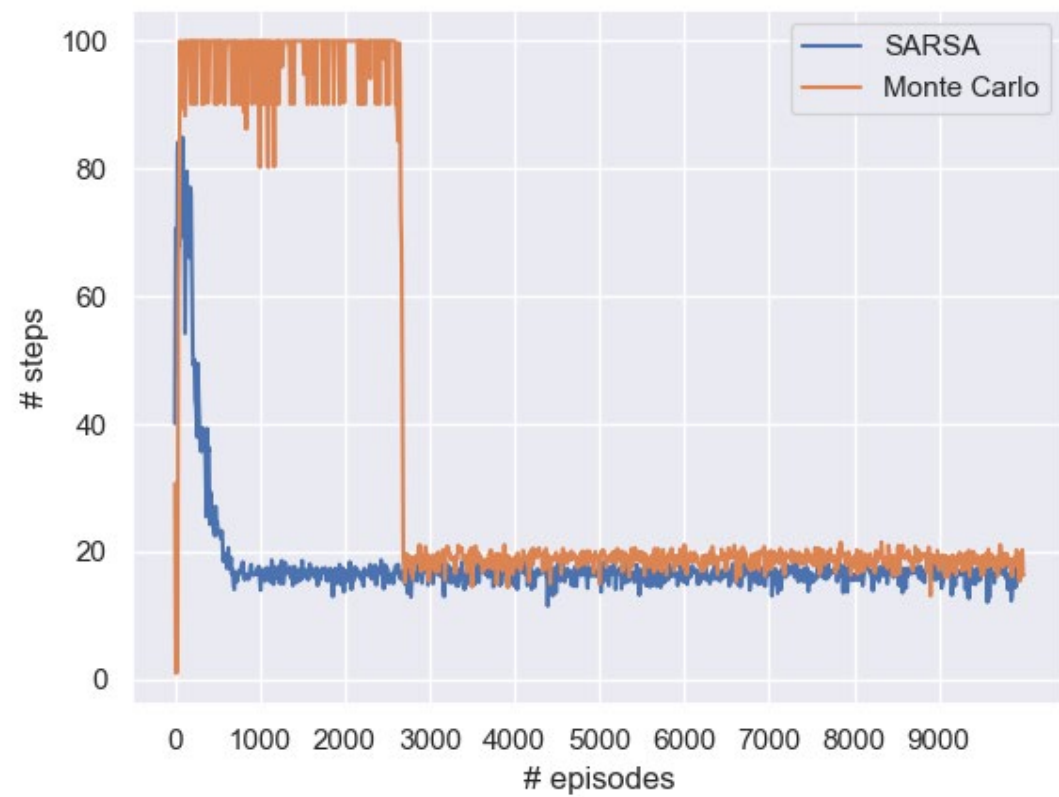
- Update value functions based purely on observed rewards

$$V(s_t) = V(s_t) + \alpha(G_t - V(s_t))$$

- Method known as **Monte Carlo learning**, double pass, backwards pass, or Temporal Difference 1/TD(1) learning







Monte Carlo Reinforcement Learning

- Main problem of Monte Carlo learning → high variance!
 - Reward trajectory may vary a lot
 - The longer the trajectory, the higher the variance
 - Convergence may take very long
- Also benefits:
 - Unbiased estimate, working only with observed rewards
 - Rewards can be backpropagated more quickly
- **Is there an alternative?**



Monte Carlo Reinforcement Learning - Summary

Monte Carlo Learning

- On-policy method
- Unbiased estimate of Q-values
- Rewards backpropagate directly through full trajectory
- High reward variance sample in lengthy trajectories



SARSA and Q-learning

Temporal Difference Learning

- Monte Carlo learning suffers from high variance
- Solution: break down updates into smaller steps
 - Every time step gives us a reward and thus some information
 - We don't need a full trajectory, however, we then need to bootstrap...
- Q-learning and SARSA are bootstrapping methods that update after every time step



TD(0) learning

- Q-learning and SARSA are **bootstrapping** methods
 - We use one estimate to update another estimate

$$[r_{t+1} + \gamma \underbrace{Q(s_{t+1}, a_{t+1})}_{\text{Estimate}}] - \underbrace{Q(s_t, a_t)}_{\text{Estimate}}$$

- Known as **temporal difference or TD(0)** approach
 - Take time step ('temporal') and compute error ('difference')



SARSA and Q-learning – Temporal Difference Learning

- Basic value approximation algorithms: SARSA and Q-learning
- Both are based on lookup tables
- Known as **temporal difference methods**
 - Take time step ('temporal') and compute error ('difference')
 - Bootstrapping methods: use one expectation to update another
- Vanilla Q-learning and SARSA address outcome space, but not state space and action space



SARSA and Q-learning – Decision-making (training)

- Decision-making mechanism:

$$a = \begin{cases} \max_{a \in A(s)} r(s, a) + Q(s, a) & \text{if } \epsilon > 0.05 \\ a = \text{random} & \text{if } \epsilon \leq 0.05 \end{cases}$$

- Most of the time, we take action with highest expected value
 - Early Q-values might be completely off!
 - Initialization matters for training behavior
- Sampled variable $\epsilon \in [0,1]$ governs exploration
 - Some exploration desirable to avoid local optima
 - Common exploration rate: 0.05 or 0.1



SARSA and Q-learning – Decision-making (application)

- **Offline learning:** learn policy in advance, then deploy
- **Online learning:** learn policy on-the-fly, incurring real rewards
- Offline learning: $\max_{a \in A(s)} r(s, a) + Q(s, a)$
 - Omit exploration after learning policy
 - May favor Q-learning → Exploration is cheap
- Online learning : same as during training
 - Learn while using the policy!
 - May favor SARSA → risk-averse approach



SARSA and Q-learning – Update rule

- **SARSA:** State, **A**ction, **R**eward, State, **A**ction, ...

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

- **Q-learning**

$$Q(s_t, a_t)$$

$$= Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a' \in \mathcal{A}(s_t)} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

- Key difference: maximization step



Exploration-exploitation dilemma

[1/2]

- All downstream values $Q(s, a)$ are unknown at the start
 - Often initialized to encourage exploration
 - E.g., very high reward or negative costs
- Epsilon-greedy algorithm
 - Take random action with probability ϵ (e.g., $\epsilon = 0.05$)
 - Take 'optimal' action with probability $1 - \epsilon$
 - Without exploration, we often get stuck into **local optima**



Exploration-exploitation dilemma

[2/2]

- Exploration-exploitation dilemma
 - **Exploitation**: better learn values of good state-action pairs
 - **Exploration**: visit unobserved areas of solution space
- The more complex and computationally intensive, the more important this dilemma gets



SARSA and Q-learning – Differences

- Key difference: maximization step
- Q-learning is off-policy, SARSA is on-policy
 - **On-policy**: Same policy is used for exploration and for updates
 - **Off-policy**: Exploration with randomness, but updates follow best actions
- Q-learning may be denoted by 'SARS'
 - May take suboptimal action a_t now, but proceed as if following best policy



SARSA algorithm

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Loop for each step of episode:

 Take action A , observe R, S'

 Choose A' from S' using policy derived from Q (e.g., ε -greedy)

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

 until S is terminal



Q-learning algorithm

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Take action A , observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$

 until S is terminal



Update procedure

- Error term: $[observation - prediction] = [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})] - Q(s_t, a_t)$
- $Q(s_t, a_t) \rightarrow$ value prediction for $t+1$ at t
- $[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})] \rightarrow$ 'observed' prediction for $t+1$ at t
 - Of course, $Q(s_{t+1}, a_{t+1})$ is **also** an estimate (bootstrap)
- Error is used to update $Q(s_t, a_t)$
 - Learning rate α dictates weight of new observation
 - Algorithm converged if $Q(s_t, a_t) = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}), \forall s_t, a_t$



Update procedure – Simplified

$$Q_T = 0$$

$$Q_{T-1} = \mathbb{E}(r_T)$$

$$Q_{T-2} = \mathbb{E}(r_{T-1} + r_T)$$

...

$$Q_0 = \mathbb{E}(r_1 + r_2 + \dots + r_{T-1} + r_T)$$

Each Q-value is estimate of trajectories















SARSA and Q-learning

- Cliff walk example (Sutton & Barto)
 - Cost -1 per step, cost -100 when falling off the cliff
 - Optimal solution is obvious (minimal number of steps)
 - With random actions, you will sometimes walk off the cliff.
- Intuitive difference:
 - Explorative action in Q-learning → limited downstream effect
 - Explorative action in SARSA → strong downstream effect
 - For small epsilon, converges to same policies



Cliff walking problem

Cliff walking problem (Sutton & Barto, 2019)

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	-100 	-100 	-100 	-100 	-100 	-100 	-100 	-100 	-100 	-100 	+10 



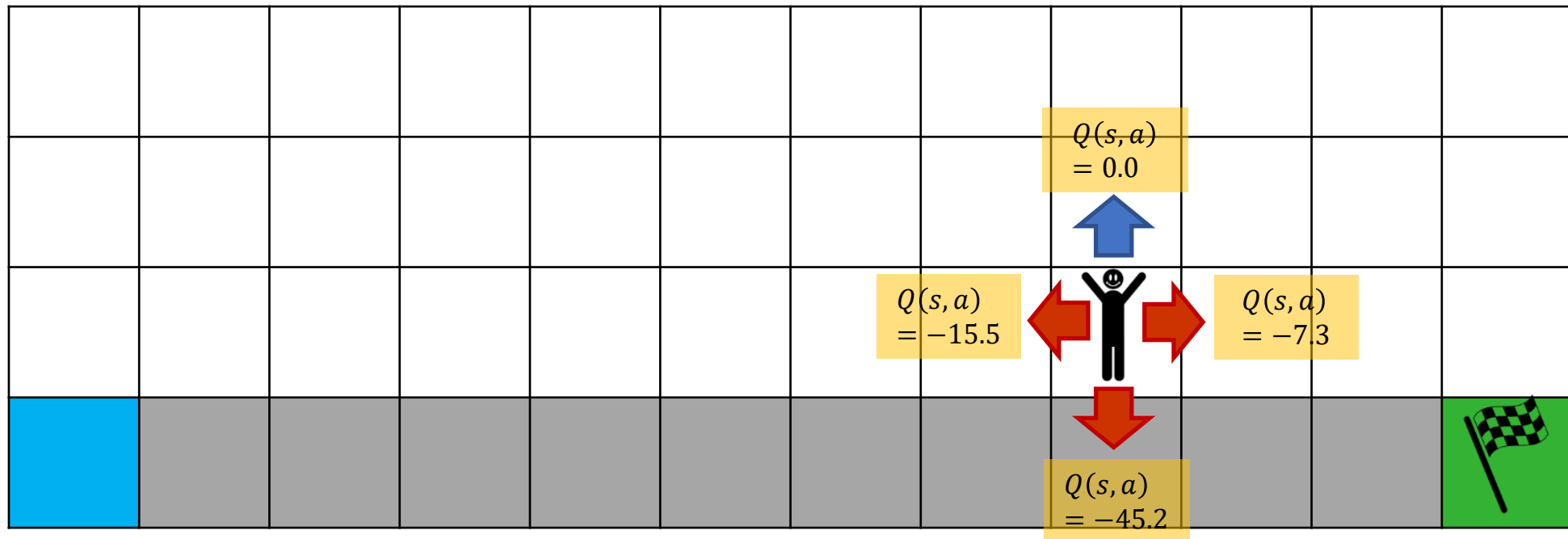
Cliff walking problem – Decision-making [1/2]

- Suppose we initialize $Q(s, a) = 0$ for all state-action pairs
- Decision at each tile \rightarrow pick action with highest Q-value
 - Early on, we have an incentive to visit unseen states
 - However, agent may get 'stuck' without ever reaching target
- Exploration:
 - With probability ϵ , pick random action
 - Often needed to avoid local optima (e.g, corner point)

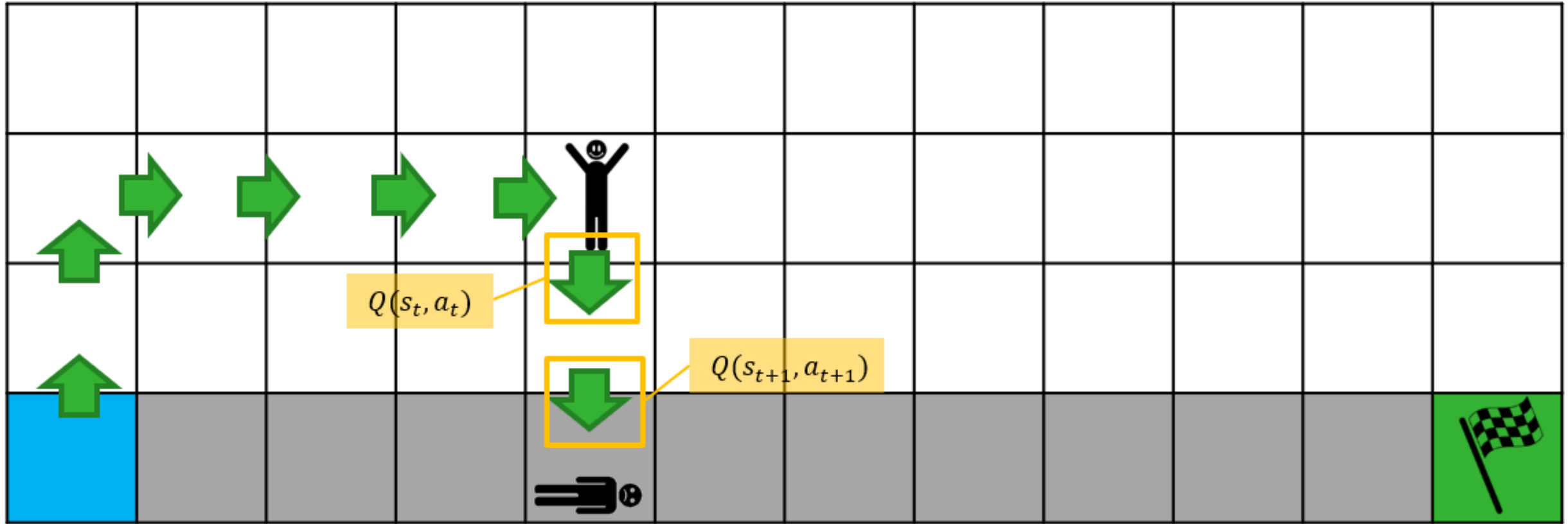


Cliff walking problem – Decision-making [2/2]

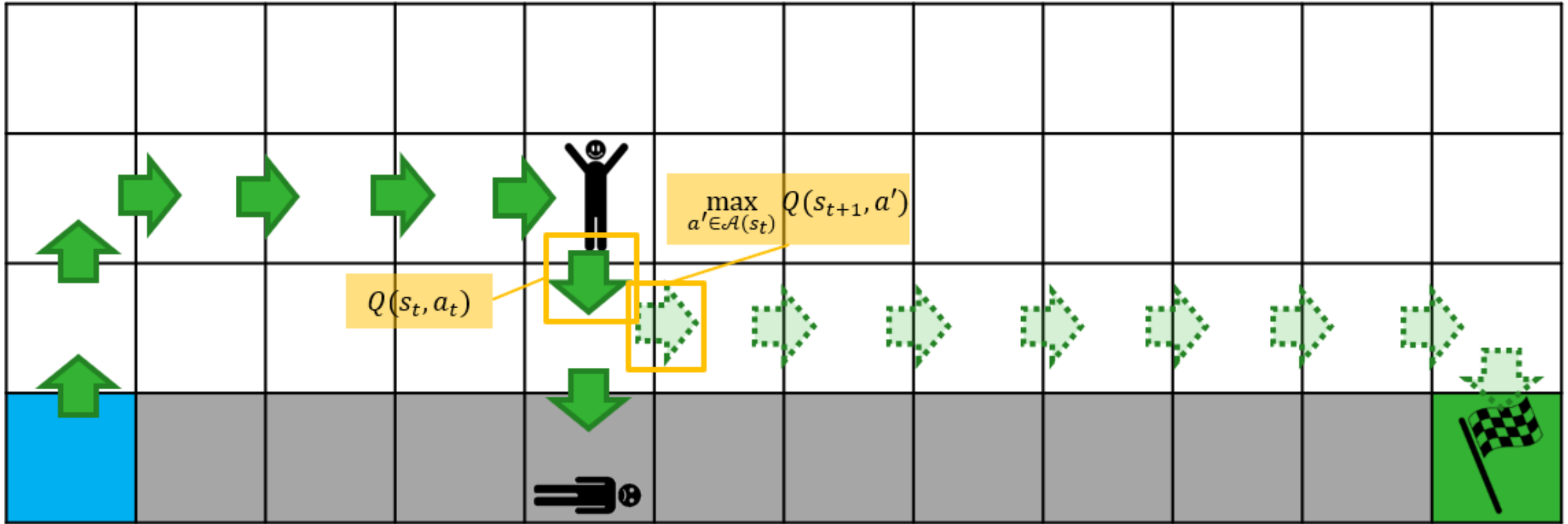
- Select action with highest Q-value (with probability $1 - \epsilon$)
- Select random action (with probability ϵ)



SARSA (on policy)



Q-learning (off policy)



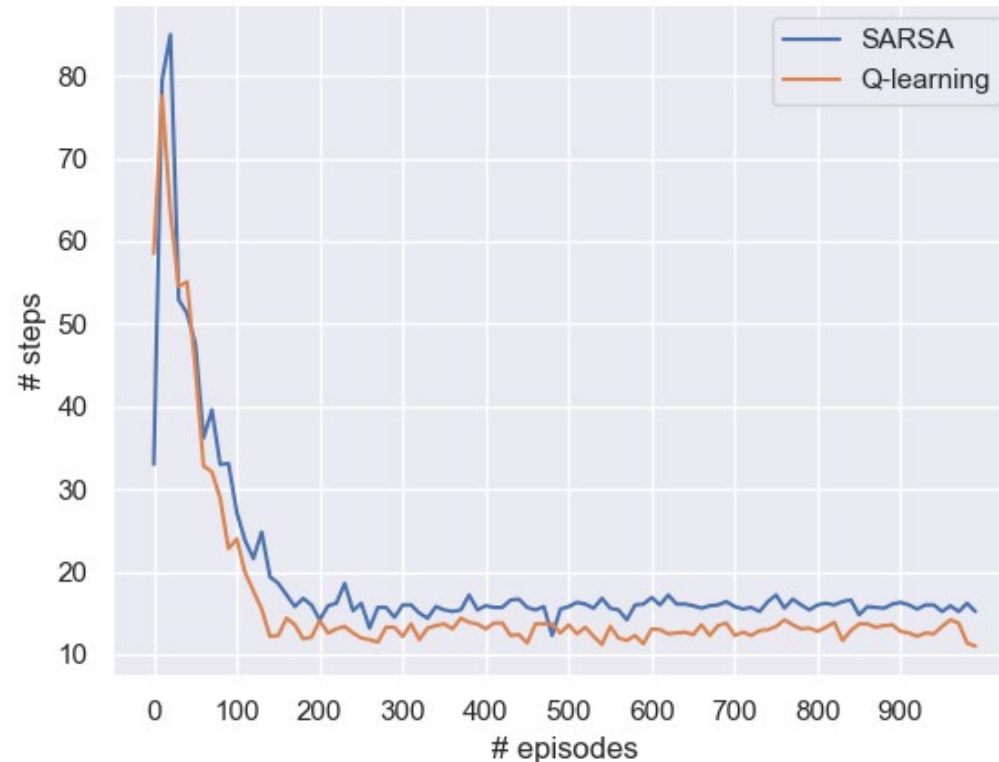
SARSA and Q-learning [1/4]

- Why does SARSA agent take a detour?



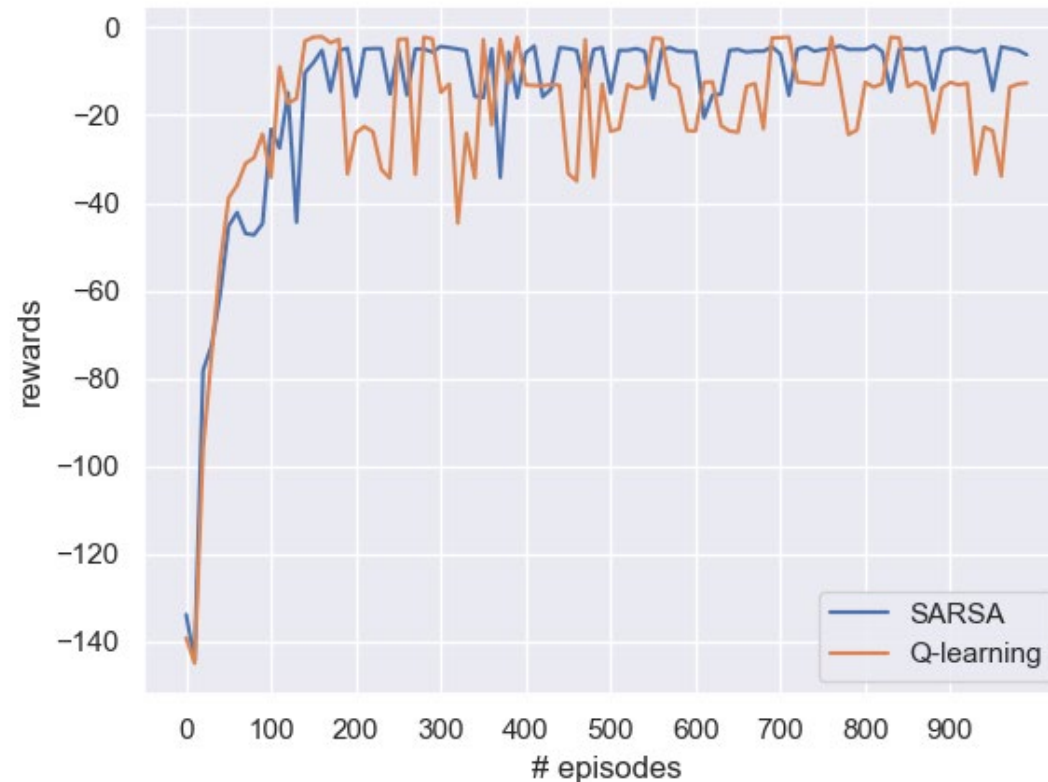
SARSA and Q-learning [2/4]

Why does Q-learning require fewer steps?



SARSA and Q-learning [3/4]

Why is SARSA reward higher?



SARSA and Q-learning [4/4]

Q-learning

- Off-policy method
- Directly learns optimal policy
- Exploration penalties mostly ignored
- Requires solving maximization function
- Higher variance per sample

SARSA

On-policy method

Learns suboptimal policy

Exploration may be heavily penalized

No internal maximization function

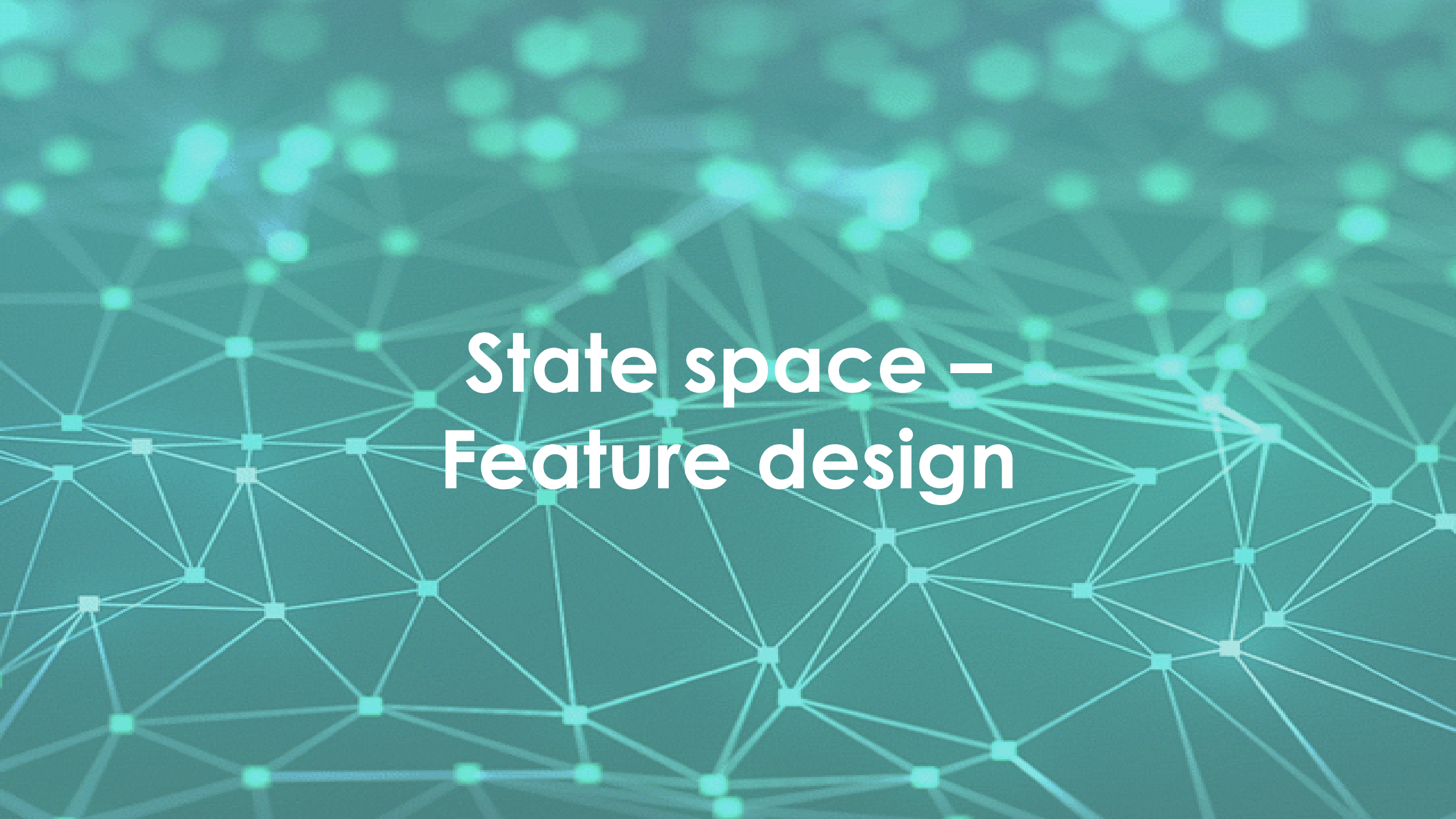
Epsilon should be tunable to converge



Problems with lookup tables

- Outcome dimension 'resolved', but not out of the woods yet
- Each state-action pair needs a value
 - Size quickly explodes: e.g., 10,000 states * 10,000 actions = 100,000,000 entries in lookup table
- Every state-action observations gives one data point, typically many observations are needed.
 - Suppose you need 100 data points per pair → 10,000,000,000 observations!
- Lookup table does not scale w.r.t. state- and action spaces





State space – Feature design

Curse II: State space

- Often impossible to visit all states $s \in S$
 - Also, lookup tables would simply get too big to store
- Recall, a state contains information needed for decision
 - Learning value function $V(s)$ helps to make good decisions
 - Optimal value functions equal optimal policy
 - However, value functions are not an objective in itself
- We approximate information in state, so we don't need to sample all of them.



Similarity of states

- Solution direction → States often have similar properties:
 - Many containers means full barge, regardless of destination
 - Most stocks tend to go up in bull markets
 - Low inventories correlate to high stockouts
- In short: viewing a state may tell us something about other states!



State space – Features

- To capture essence of a state, we can design **explanatory variables** (called **features** in machine learning)
- Linear regression model
- Polynomials capture many effects (e.g., higher order interactions)
- Neural networks can transform features to value
- Linear variant:

$$\tilde{V}(s_t) = \min_{a_t \in A(s_t)} C(s_t, a_t) + \sum_{f \in F} \theta_f(s_t, a_t) \cdot \phi_f$$

$$\sum_{f \in F} \theta_f(s_t, a_t) \cdot \phi_f = \theta_1(s_t, a_t) \cdot \phi_1 + \theta_2(s_t, a_t) \cdot \phi_2 + \dots + \theta_{|F|}(s_t, a_t) \cdot \phi_{|F|}$$

Credit rating

Learned weight



State space – Basis functions

- The challenge is to extract relevant **features** from state
 - In regression, these are simply explanatory variables.
 - Features are extracted using basis functions θ_f
 - $\phi_f: S \rightarrow \mathbb{R}$ ▶ Basis function, extracting feature f
 - $\phi = [\phi_1, \phi_2, \dots, \phi_{|F|}]$
 $f \in F$ ▶ Vector of basis functions, returning all features
- Examples:
 - $\theta_f: [3, 2, 1, 5, 0, 3, 1, 1, 0] \rightarrow 16$ ▶ # containers
 - $\theta_f: [3, 2, 1, 5, 0, 3, 1, 1, 0] \rightarrow 9^2$ ▶ # red containers squared
 - $\theta_f: [3, 2, 1, 5, 0, 3, 1, 1, 0] \rightarrow \frac{6}{16}$ ▶ Ratio urgent containers



State space – Neural network

- Conceptually similar to regression model:

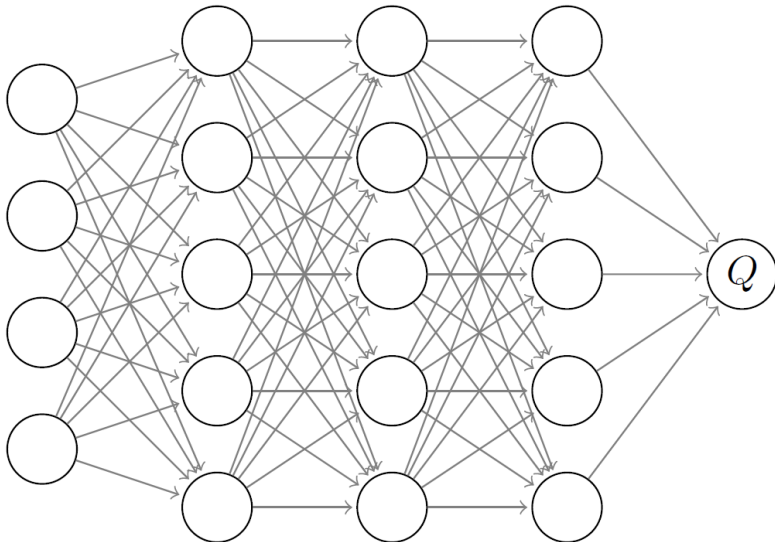
$$\tilde{V}(s_t) = \min_{a_t \in A(s_t)} \mathcal{C}(s_t, a_t) + N(\phi_f(s_t, a_t), \theta_f)$$

Neural network

Input layer

Network weights

Input layer Hidden layer 1 Hidden layer 2 Hidden layer 3 Output layer



State space – Finding features [1/4]

- Often good intuitive understanding of problem
- Container shipping
 - Consolidating containers for same destination beneficial
 - Prioritizing nearest due dates makes sense
 - With few containers, waiting might increase efficiency
- Portfolio optimization
 - Difference between stock price and long-term average
 - Correlation between stocks



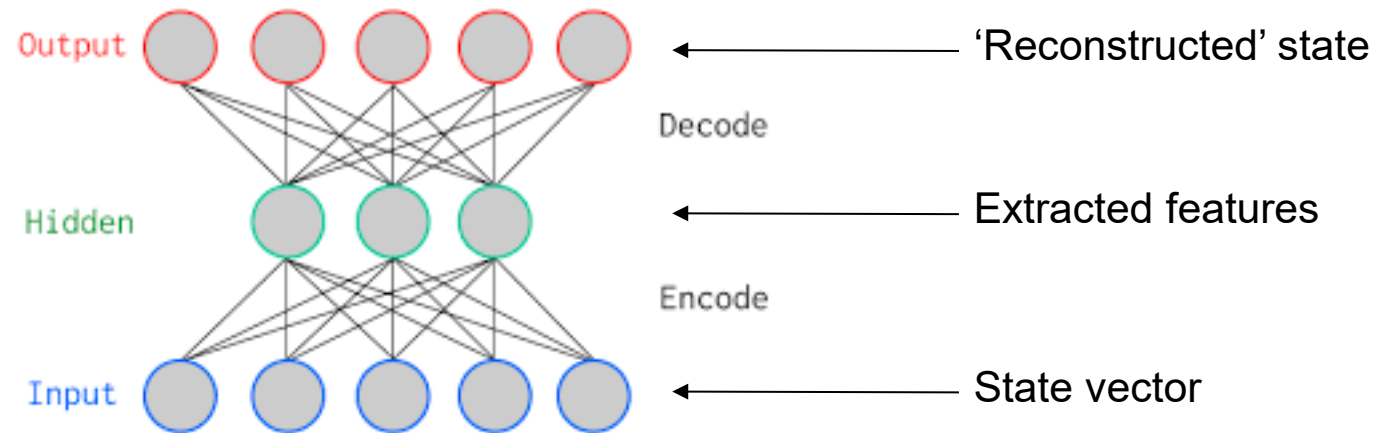
State space – Finding features [2/4]

- Manual feature design requires domain knowledge and good understanding of problem structure
 - Often also complicated interactions between factors
 - Ratio between urgent containers and total containers
 - Nonlinear relations
 - Squared price
- Finding a good set of features involves trial-and-error
 - Measure policy quality for various sets
 - Compute R^2 for explanatory value of sets



State space – Finding features [3/4]

- Automatic feature extraction:
 - Auto-encoder learns features from state
 - If you can reconstruct state, features are informative



Source: <https://commons.wikimedia.org/wiki/File:AutoEncoder.png>



State space – Finding features [4/4]

- Aggregation into higher dimension
- Some problems have natural hierarchical structure
 - Trade detail for generalization
- Example:
 - Aggregate addresses to zip code
 - Aggregate zip codes to cities
 - Aggregate cities to regions



Nederland

Nederland



We may not need *all* information embedded in the state to make good decisions!

From national perspective,
 $V(\text{Hengelo}) \approx V(\text{Enschede})$



Action space – Heuristics and mathematical programming

Course III: Action space

- Picking best actions requires evaluation all actions $a \in \mathbf{A}$
- Least studied curse of dimensionality
 - Typical engineering- and computer science problems have relatively few actions
 - Chess: extreme number of board positions, but number of actions per state is enumerable
 - Video game: Mario can run left, right, jump, etc.
 - Robot arm: control a few joints, e.g., $a = [a_{\text{shoulder}}, a_{\text{elbow}}, a_{\text{wrist}}]$
- However, real-world problems often entail vast decision spaces



Solution approaches

- Enumeration of actions might be possible
 - Problem transformed into simple **deterministic problem**
- Mathematical programming
 - Extremely powerful for high-dimensional problems
 - Exploits convex problem structure
 - Preserves optimality!
- Heuristics
 - Often 'good' solutions within reasonable time



Heuristic approaches

- Action space reduction
 - Cut non-promising regions, e.g.:
 - No barges with less than 50% capacity utilization
 - Don't deplete battery below 30%
 - Don't buy stocks above long-term average
- Deploy existing heuristics extended with features, e.g.:
 - Classic VRP heuristic with downstream cost component
- Tailored meta-heuristics might be necessary



Mathematical programming [1/2]

- Many formulations readily exist
 - Simply expand with $+\sum_{f \in F} \theta_f \phi_f$ to incorporate downstream values
 - Depending on problem structure, massive problems may be solved
 - Remember we must perform thousands of iterations, so at most a few seconds per decision
- For *linear* programs, features θ_f must be linear expressions as well
 - Can be quite challenging



Mathematical programming [2/2]

- ILPs can get messy...
- Post-decision variables not always easy to define
- May require many help variables
- Artificial constructions to linearize features
- Markowitz portfolio optimization
- Quadratic program
 - Quadratic objective with linear constraints
 - Can be expanded with linearized features

$$\begin{aligned}
 \min_{x_i \in \mathcal{X}(S_i)} & \left(\sum_{q^w \in \mathcal{Q}^w} z_{q^w}^w \cdot q^w \cdot e^{w,fix} + \sum_{q^w \in \mathcal{Q}^w} z_{q^w}^w \cdot q^w \cdot e^{w,fix} + \right. \\
 & \sum_{n \in \mathcal{N}} \sum_{d \in \mathcal{D}} \sum_{q^w \in \mathcal{Q}^w} \sum_{q^w \in \mathcal{Q}^w} z_{n,d,q^w,q^w} \cdot e_{n,d,q^w,q^w}^{w,w} + \\
 & \left. \sum_{v \in \mathcal{V}} z_v^{fix} \cdot e_v^{fix} + \sum_{f \in \mathcal{F}} \theta_{f,1} \cdot \phi_f(S_f^1) \right), \\
 & \sum_{v \in \mathcal{V}} \sum_{f \in \mathcal{F}} \sum_{t \in \mathcal{T}} (I_{v,f,t,0,t} - x_{v,f,t,0,t}) \leq t^{max}, \quad (3.7.1) \\
 & x_{v,f,t,1,t} \cdot \mu \leq I_{v,f,t,1,t}, \quad (3.7.2) \\
 & \forall (v,f,t) \in \mathcal{V} \times \mathcal{L} \times \mathcal{T}^1, \quad x_{v,f,t,0,0} = I_{v,f,t,0,0}, \quad (3.7.3) \\
 & \forall (v,f) \in \mathcal{V} \times \mathcal{L}, \quad x_{v,f,t,1,t} \cdot \mu = 0, \quad (3.7.4) \\
 & t^0 > 0, \forall (v,f,t) \in \mathcal{V} \times \mathcal{L} \times \mathcal{T}^1, \quad I_{v,f,t,1,t}^0 = I_{v,f,t,1,t} - x_{v,f,t,1,t}, \quad (3.7.5) \\
 & \forall (v,f,t) \in \mathcal{V} \times \mathcal{L} \times \mathcal{T}^1, \\
 & z_{\tau_r}^{max} \geq \frac{\sum_{n \in \mathcal{N}} \sum_{d \in \mathcal{D}} \sum_{q^w \in \mathcal{Q}^w} \sum_{q^w \in \mathcal{Q}^w} (z_{n,d,q^w,q^w} \cdot s_{n,d,q^w,q^w}) - \tau_r}{\tau_r \cdot MaxRoute}, \quad (3.7.6) \\
 & \forall \tau_r \in \{0, \dots, MaxRoute\}, \\
 & \tau_r \in \{0, \dots, MaxRoute\} \Rightarrow z_{\tau_r}^{max} = \sum_{n \in \mathcal{N}} \sum_{d \in \mathcal{D}} \sum_{q^w \in \mathcal{Q}^w} \sum_{q^w \in \mathcal{Q}^w} z_{n,d,q^w,q^w} \cdot s_{n,d,q^w,q^w}, \quad (3.7.7) \\
 & \forall \tau_r \in \{0, 1, \dots, \tau^{MaxRoute}\}, \quad q_{\tau_r}^{fix} \leq q_{\tau_r}^w, \quad (3.7.8) \\
 & q_{\tau_r}^{fix} \leq q_{\tau_r}^w - \sum_{q^w \in \mathcal{Q}^w} z_{q^w}^w \cdot q^w + (1 - z_{\tau_r}^{max}) q_{\tau_r}^{max}, \quad (3.7.9) \\
 & \forall \tau_r \in \{0, 1, \dots, \tau^{MaxRoute}\}, \quad q_{\tau_r}^{fix} \geq q_{\tau_r}^w - \sum_{q^w \in \mathcal{Q}^w} z_{q^w}^w \cdot q^w, \quad (3.7.10) \\
 & \forall \tau_r \in \{0, 1, \dots, \tau^{MaxRoute}\}, \quad \sum_{q^w \in \mathcal{Q}^w} z_{q^w}^w \cdot q^w + \sum_{q^w \in \mathcal{Q}^w} z_{q^w}^w \cdot q^w \geq \sum_{f \in \mathcal{F}} \sum_{t \in \mathcal{T}} \sum_{v \in \mathcal{V}} x_{v,f,t,1,t} \cdot \mu, \quad (3.7.11) \\
 & \sum_{v \in \mathcal{V}} \sum_{f \in \mathcal{F}} \sum_{t \in \mathcal{T}} d_{0,v} \leq q_{\tau_r}^{fix} \cdot d_{0,v} \quad \forall d \in \mathcal{D}, \quad (3.7.12) \\
 & \sum_{d \in \mathcal{D}} d_{0,v}^0 = 1, \quad (3.7.13) \\
 & z_{q^w}^{max} \geq z_{q^w}^w, \quad \forall q^w \in \mathcal{Q}^w \setminus \emptyset, \quad (3.7.14) \\
 & \sum_{q^w \in \mathcal{Q}^w} z_{q^w}^w = 1, \quad (3.7.15) \\
 & \sum_{q^w \in \mathcal{Q}^w} z_{q^w}^w = 1, \quad (3.7.16) \\
 & z_v^{fix} \geq \frac{\sum_{n \in \mathcal{N}} \sum_{d \in \mathcal{D}} \sum_{q^w \in \mathcal{Q}^w} \sum_{q^w \in \mathcal{Q}^w} x_{n,d,q^w,q^w} \cdot \mu}{\sum_{f \in \mathcal{F}} \sum_{t \in \mathcal{T}} \sum_{v \in \mathcal{V}} x_{v,f,t,1,t} \cdot \mu}, \quad \forall v \in \mathcal{V}, \quad (3.7.17) \\
 & z_v^{fix} \leq \frac{\sum_{n \in \mathcal{N}} \sum_{d \in \mathcal{D}} \sum_{q^w \in \mathcal{Q}^w} \sum_{q^w \in \mathcal{Q}^w} x_{n,d,q^w,q^w} \cdot \mu}{\sum_{f \in \mathcal{F}} \sum_{t \in \mathcal{T}} \sum_{v \in \mathcal{V}} x_{v,f,t,1,t} \cdot \mu}, \quad \forall v \in \mathcal{V}, \quad (3.7.18) \\
 & \sum_{n \in \mathcal{N}} z_n^{max} = \sum_{v \in \mathcal{V}} z_v^{fix}, \quad (3.7.19) \\
 & \sum_{n \in \mathcal{N}} z_n^{max} = 1, \quad (3.7.20) \\
 & z_{n,d,q^w,q^w} \leq z_n^{max}, \quad (3.7.21) \\
 & \forall (n,d,q^w,q^w) \in \mathcal{N} \times \mathcal{D} \times \mathcal{Q}^w \times \mathcal{Q}^w, \quad z_{n,d,q^w,q^w} \leq z_{q^w}^w, \quad (3.7.22) \\
 & \forall (n,d,q^w,q^w) \in \mathcal{N} \times \mathcal{D} \times \mathcal{Q}^w \times \mathcal{Q}^w, \quad z_{n,d,q^w,q^w} \leq z_{q^w}^w, \quad (3.7.23) \\
 & \forall (n,d,q^w,q^w) \in \mathcal{N} \times \mathcal{D} \times \mathcal{Q}^w \times \mathcal{Q}^w, \quad z_{n,d,q^w,q^w} \leq z_{q^w}^w, \quad (3.7.24) \\
 & \forall (n,d,q^w,q^w) \in \mathcal{N} \times \mathcal{D} \times \mathcal{Q}^w \times \mathcal{Q}^w, \quad z_{n,d,q^w,q^w} \geq z_{q^w}^w + z_{q^w}^w + z_{q^w}^w - 3, \quad (3.7.25) \\
 & \forall (n,d,q^w,q^w) \in \mathcal{N} \times \mathcal{D} \times \mathcal{Q}^w \times \mathcal{Q}^w, \\
 & z_{\tau_r}^{max} \in \{0,1\} \quad \forall \tau_r \in \{0, \dots, MaxRoute\}, \\
 & z_{q^w}^w \in \{0,1\} \quad \forall q^w \in \mathcal{Q}^w, \\
 & z_n^{max} \in \{0,1\} \quad \forall n \in \mathcal{N}, \\
 & z_v^{fix} \in \{0,1\} \quad \forall v \in \mathcal{V},
 \end{aligned}$$



Curses of dimensionality – To summarize

- Outcome space
 - Sample outcomes $\omega \in \Omega$
 - Unbiased estimate of outcome process
- State space
 - Represent states by set of features
 - No need to visit every state $s \in S$
- Action space
 - Solve decision problem with ILP or heuristic
 - More efficient than enumerating all actions $a \in A$





Wrapping up

Recap

- Resolve curses of dimensionality
 - **Outcome**: repeatedly sample random transitions
 - **State**: replace state by set of features for value function
 - **Action**: solve using mathematical programming or heuristics
- Lookup table approach:
 - Post-decision states and state-action pairs
 - Monte Carlo learning, Q-learning and SARSA
- Tradeoff: more 'tricks' \leftrightarrow more complexity





DIGITAL



**Funded by
the European Union**

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Horizon Europe: Marie Skłodowska-Curie Actions. Neither the European Union nor the granting authority can be held responsible for them.

This project has received funding from the European Union's Horizon Europe research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101119635



DIGITAL