Introduction to Reinforcement Learning

Deep Learning IndabaX Sudan

Dr. Ndivhuwo Makondo

Research Scientist: Artificial Intelligence

Visiting Lecturer: School of Computer Science and Applied Mathematics, the University of the Witwatersrand

Research Associate: RAIL Lab (https://www.raillab.org/)

Email: ndivhuwo9@gmail.com

Outline

1. **Speaker Bio**
2. **Previous Work**
3. **Intro to RL**
4. **Multi-armed Bandits**
5. **Markov Decision Processes**
6. **Application to Malaria Control**
7. **Summary**

Outline

Speaker Bio

1. **BSc. Electrical Engineering, University of Cape Town**
   - **Control Systems**
2. **MSc. Electrical Engineering, University of Cape Town**
   - **Modelling and control of robot manipulators**
   - **MSc. Studentship, Council for Scientific and Industrial Research (CSIR)**
   - **Advisors: Prof. Martin Braae, Jonathan Claassens, Dr. Nkgatho Tlale**
3. **PhD. Computational Intelligence and System Science, Tokyo Institute of Technology (TiTECH)**
   - **Accelerating learning of motor skills with knowledge transfer**
   - **Advisors: Prof. Osamu Hasegawa and Prof. Benjamin Rosman**

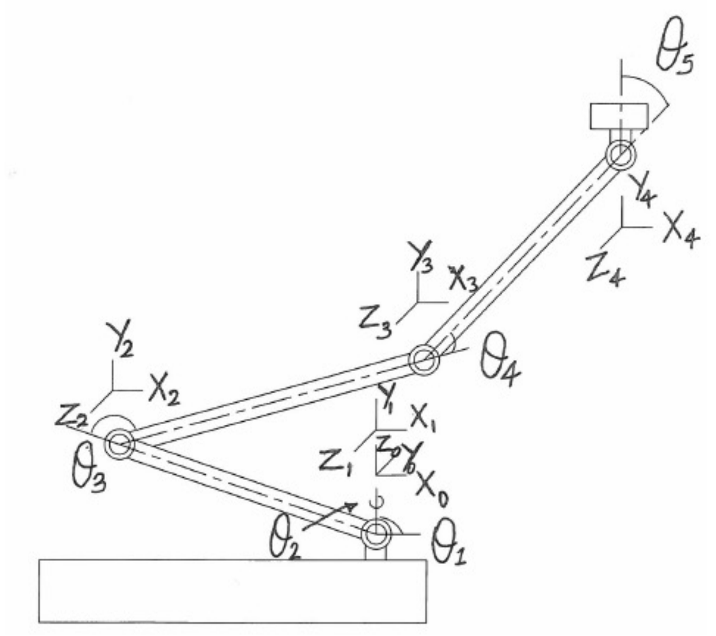
Modeling and control of a robot manipulator

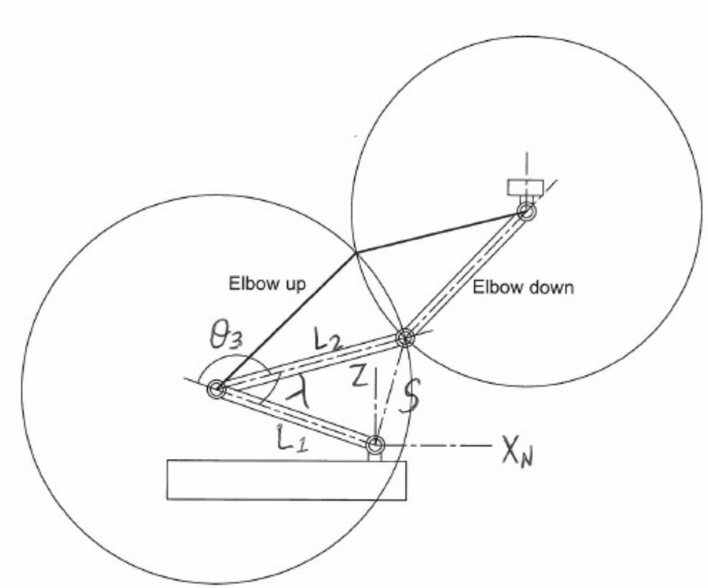
1. **Model the kinematics**
2. **Motion Planning**
3. **Avoid tip-over motions**

Outline

Modeling and control of a robot manipulator

1. **Geometrically model the kinematics**
   - **Closed-form solution**
   - **Specific to robot**
2. **Numerical models**
   - **General**
   - **Iterative solutions**
3. **General frameworks:**
   - **URDF: https://wiki.ros.org/urdf**
   - **Kinematics and Dynamics Library (KDL):** **https://www.orocos.org/kdl.html**
   - **ROS: https://www.ros.org/**





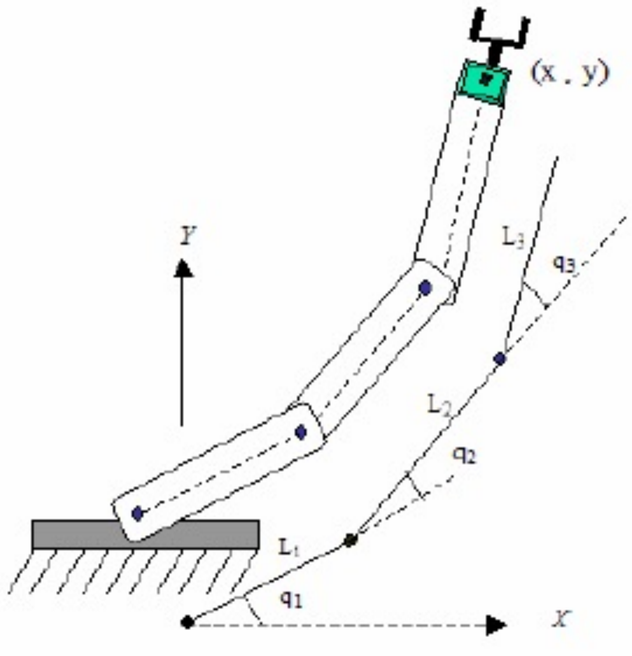
Makondo, et al., *2012*

Why learning?

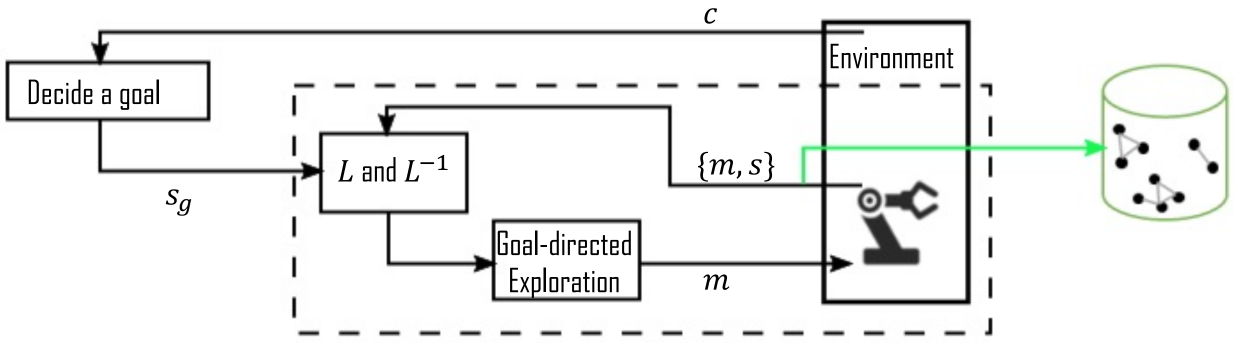
1. **Kinematics**
    - **Inaccurate robot parameters**
    - **Inaccurate sensors**
    - **Wear and tear over time**
    - **Soft robotics**
2. **Dynamics**
    - **Non-linear dynamics**
3. **Skills (low-level trajectories)**
    - **Planning can be computationally expensive for complex systems**
    - **Planning requires accurate robot models**
    - **Planning can fail to obtain complex trajectories (e.g., swinging, etc.)**

Outline

Supervised Learning (Kinematics example)



1. **Setup**
   - **Data: [(q_1,x_1),(q_2,x_2),…,(q_T,x_T)]**
   - **Model: $y = f(x)$**
   - **Loss: MSE(y,$\hat{y}$)**
   - **Techniques: kNN, Neural Network, SVR, etc.**
2. **Data collection**
   - **Developmental robotics**
   - **Goal vs Motor Babbling**
   - **Random vs Active explor**



Supervised Learning (Dynamics example)

Analytical model: Rigid Body Dynamics (RBD) model

$$\boldsymbol{\tau} = M(\boldsymbol{q})\ddot{\boldsymbol{q}}_d + C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}_d + F(\dot{\boldsymbol{q}}_d) + G(\boldsymbol{q}_d) + J(\boldsymbol{q})^T g$$

Inertia Matrix

Coriolis Matrix

Frictional Forces

Gravitational Forces

Task-space disturbances

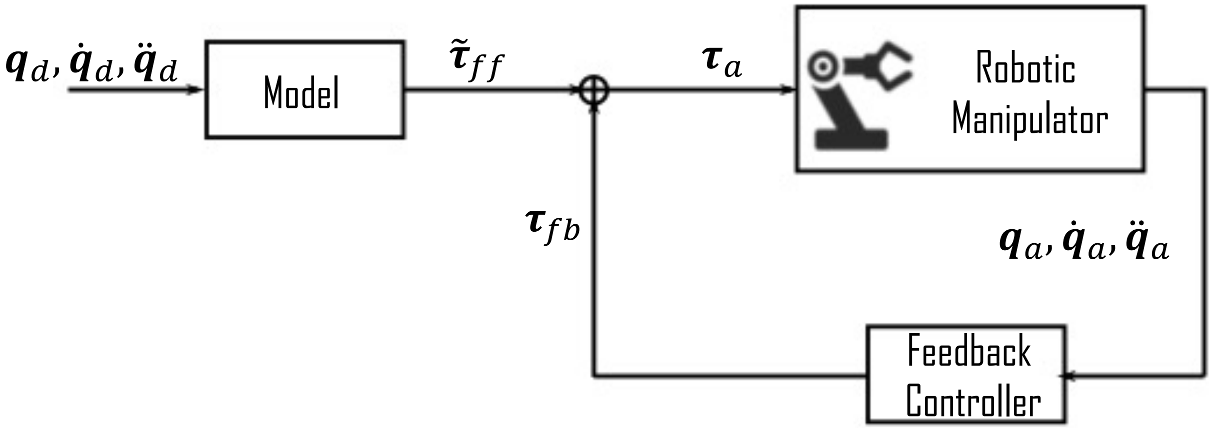
Learning-based model: Standard regression

$$\boldsymbol{\tau} = D(\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d, \ddot{\boldsymbol{q}}_d)$$

Outline

Supervised Learning (Dynamics example)

$\{\boldsymbol{q}_d, \dot{\boldsymbol{q}}_d, \ddot{\boldsymbol{q}}_d\}$: desired trajectory
$\tilde{\boldsymbol{\tau}}_{ff}$: estimated feedforward torque
$\boldsymbol{\tau}_{fb}$: feedback torque
$\{\boldsymbol{q}_a, \dot{\boldsymbol{q}}_a, \ddot{\boldsymbol{q}}_a\}$: actual trajectory
$\boldsymbol{\tau}_a$: actual torque (applied)



- **Control robot to follow desired trajectory**
  - **Learns model from data**
  - **Models unknown non-linearities**
  - **Complex, light robots**

Outline

Challenges with learning

1. **Large state and action spaces – large datasets**
   - **Robot must perform trial-and-error for long periods of time from scratch**
2. **Unsafe random exploration**
   - **Risk damaging the robot**

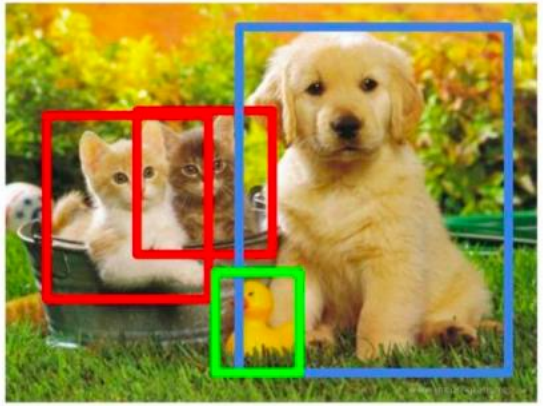
Potential solutions

1. **Using available prior knowledge**
2. **Transfer learning**
   - **Use prior knowledge generated by other similar robots**
3. **Integrating domain knowledge**
   - **Use our understanding of physics together with machine learning**
4. **Sim2real transfer**
   - **Learn on simulated robot and transfer to real robot**
5. **Meta-learning**
   - **Learning variations of the task and optimize to do well on a new related task**

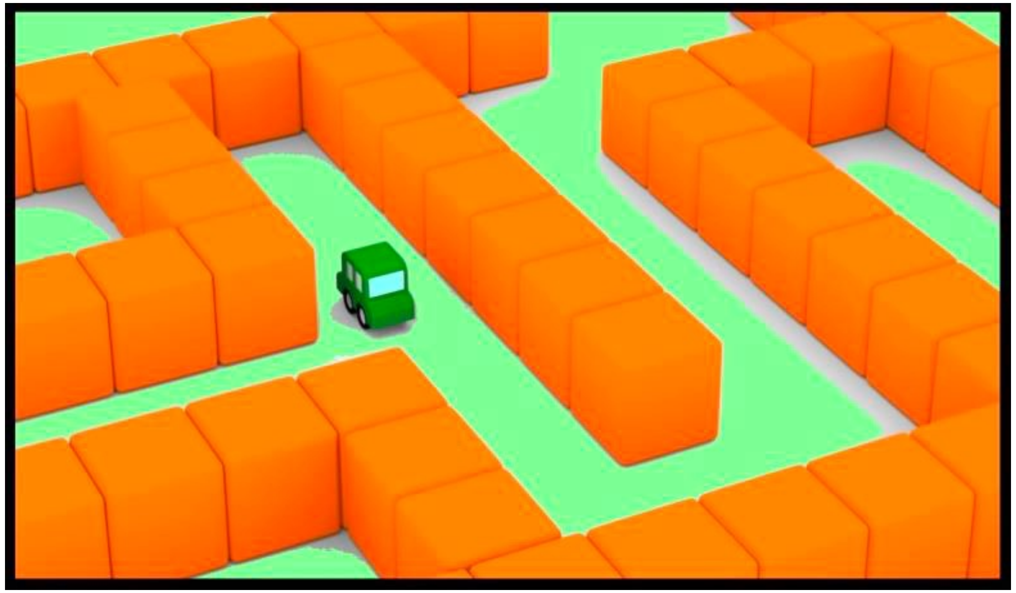
Outline

Introduction to Reinforcement Learning

1. **Supervised learning**
   - **One-shot decision, single decision point**
   - **Classify input into one of many classes**
   - **Predict torque value for current pose**
2. **Reinforcement learning**
   - **Multiple sequential decision points**
   - **Am I doing the right thing?**
   - **Present decisions impact future outcomes**
   - **Actions affect the environment!**



CAT, DOG, DUCK

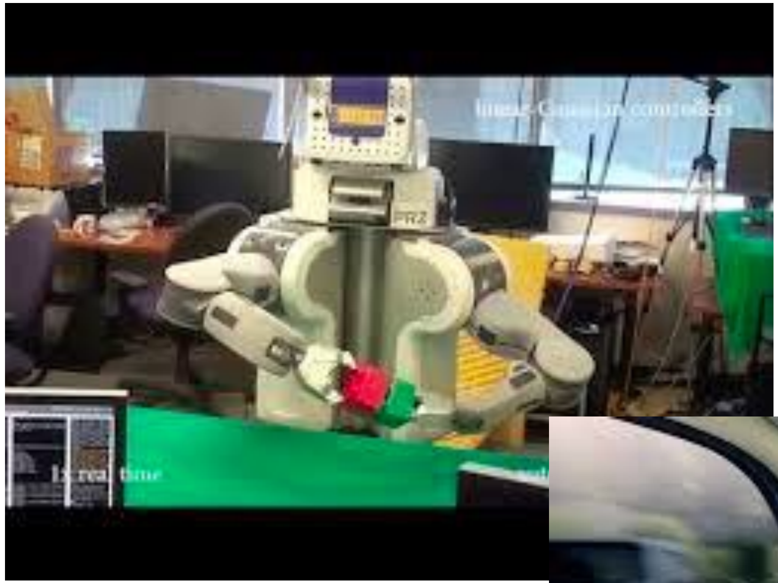


Introduction to Reinforcement Learning

1. **Why not supervised learning?**
   - **Requires labels (e.g., desired trajectories)**
   - **Desired trajectory may be unknown a-priori but feedback can be easily provided**
2. **What is reinforcement learning?**
   - **Reinforcement learning is the branch of machine learning relating to learning in sequential decision-making settings**
   - **Behavior learning with multiple decisions, long-term effects and uncertainty**
   - **Like control theory without models**

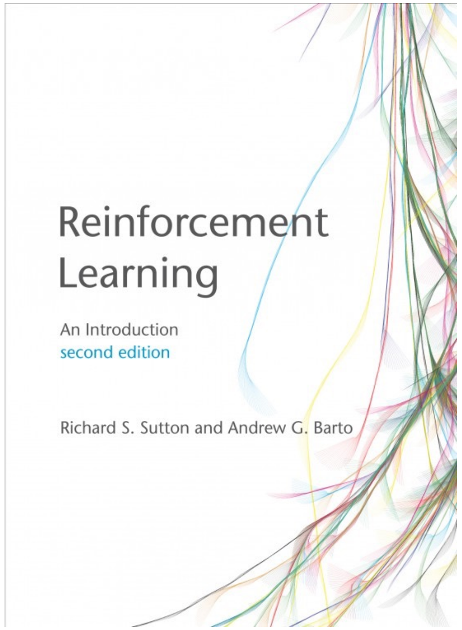
Outline

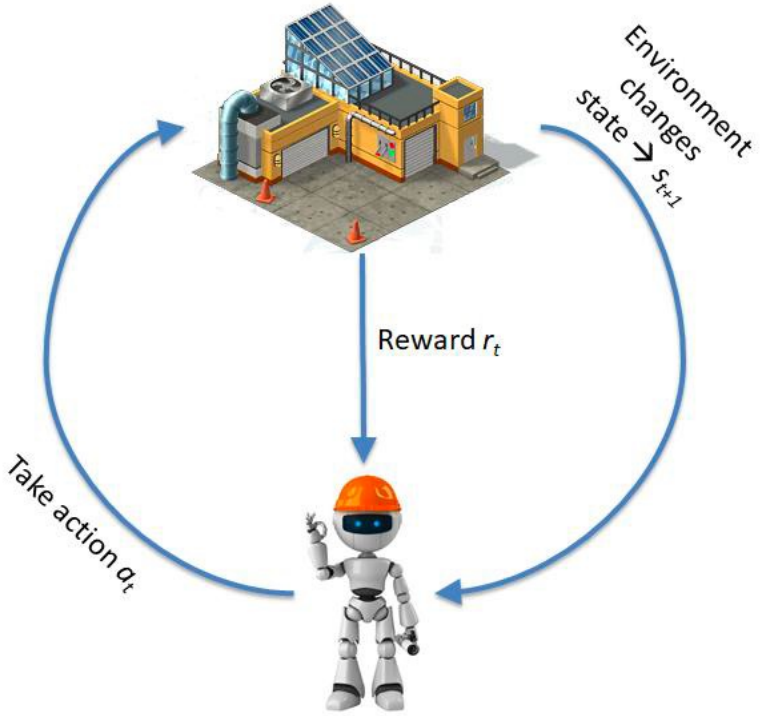
Applications



Interacting with an environment

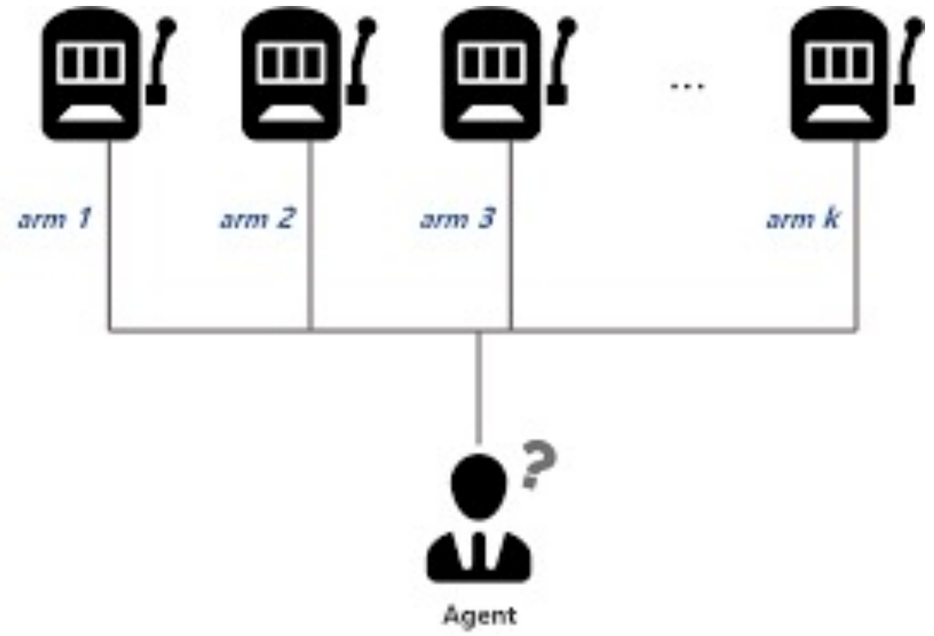
- **Decision maker (agent) exists within an environment**
- **Agent takes actions based on the environment state**
- **Environment state updates**
- **Agent receives feedback as rewards**





Multi-armed Bandits

- **No state (or a single state)**
- **Discrete actions**
- **Agent takes action A_t and receives feedback as rewards R_t at time step t.**
- **Goal: maximize total expected reward over some period of time, called episode (e.g., 1000 steps)**



Action-value Methods

Each action has an associated value $q_*(a)$

$$q_*(a) \doteq \mathbb{E}[R_t \mid A_t = a]$$

Access only to estimate at any time step ***t*****:**

$$Q_t(a)$$

Exploitation: ***greedy action*** **– exploit known values**

$$A_t \doteq \arg\max_a Q_t(a).$$

Exploration: non-greedy or random action – explore to discover new values

Learning action values:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

Incrementally:

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}. \quad = \quad Q_n + \frac{1}{n}\Big[R_n - Q_n\Big]$$

$$NewEstimate \leftarrow OldEstimate + StepSize \Big[Target - OldEstimate\Big] \qquad Q_{n+1} \doteq Q_n + \alpha\Big[R_n - Q_n\Big].$$

Action selection: ϵ -greedy

With small probability 1-ϵ select greedy action

$$A_t \doteq \arg\max_a Q_t(a)$$

With probability ϵ select non-greedy action

Can decay ϵ over time – exploration -> exploitation

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Repeat forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

Action selection: Upper Confidence Bounds (UCB)

Balance exploration vs exploitation

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Repeat forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

Gradient Bandit Methods

Each action has an associated numerical preference $H_t(a)$

The higher $H_t(a)$ the more often that action is taken

Unlike action value, preference has no interpretation in terms of reward

Relative preference is important

Policy is a soft-max distribution

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a),$$

Learning preferences after selecting A_t and receiving reward R_t by stochastic gradient ascent:

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t,$$

$\alpha > 0$ is a step size parameter and $\bar{R} \in \mathbb{R}$ is running average reward (baseline)

Contextual Bandits (Associative Bandits)

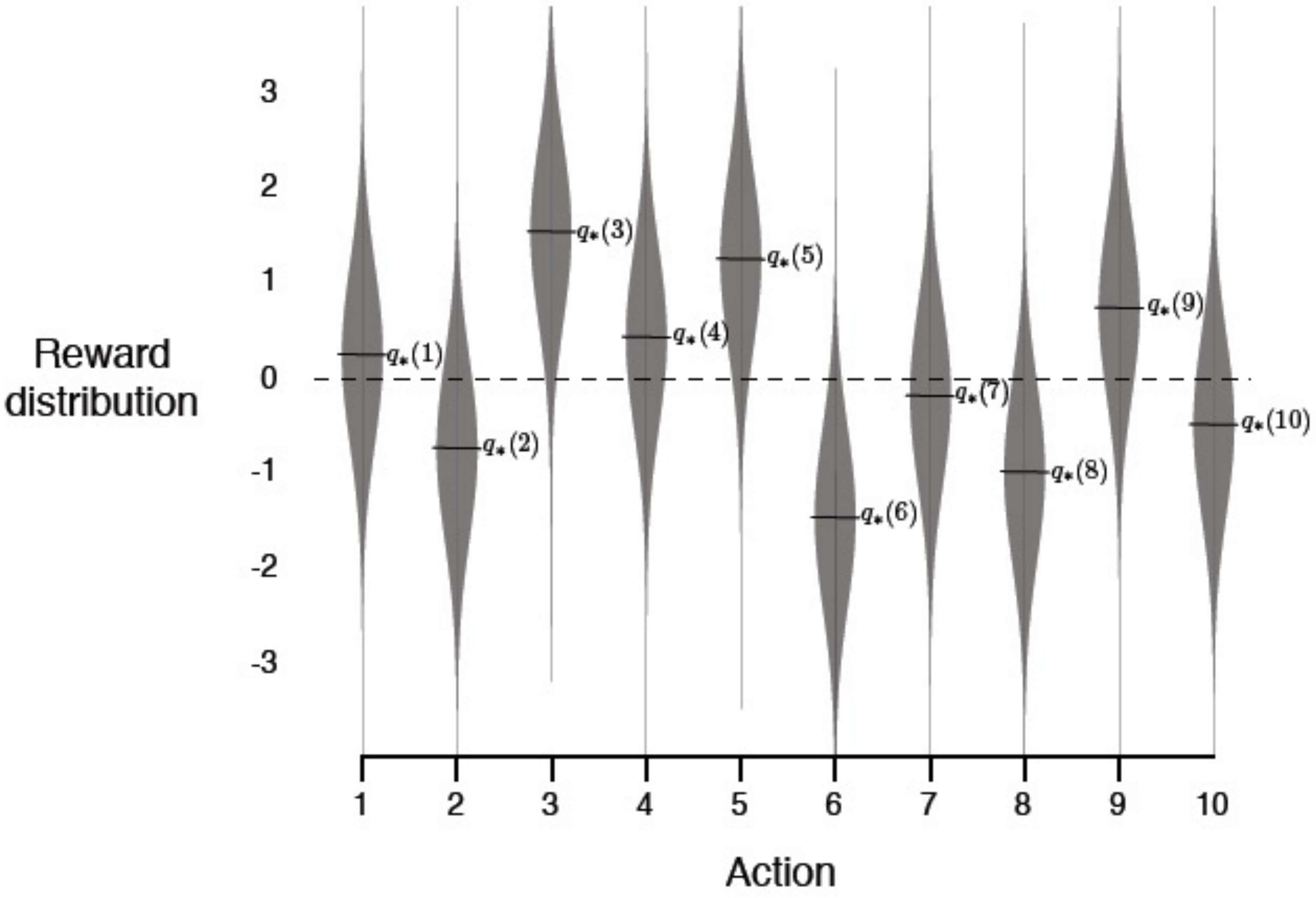
- **Independent states**
- **Discrete actions**
- **Agent takes action A_t based on associated state and receives feedback as rewards R_t at time step t.**
- **Goal: maximize total expected reward over some period of time, called episode (e.g., 1000 steps)**
- **States are presented at random and independent of actions executed.**
- **Action-value methods:**

$$Q_i(s_t, \mathbf{a}_t) = Q_{i-1}(s_t, \mathbf{a}_t) + \alpha(R_t - Q_{i-1}(s_t, \mathbf{a}_t))$$

- **Gradient Bandit methods:**

$$\pi_i(s, \mathbf{a})$$

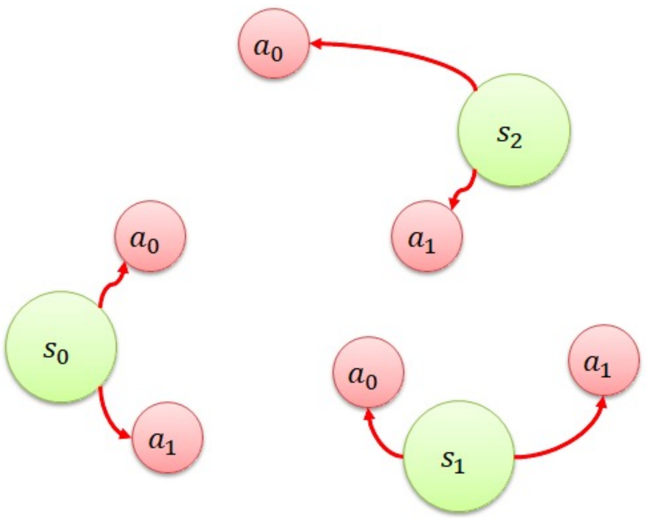
Code Walkthrough



https://github.com/bgalbraith/bandits

Markov Decision Process

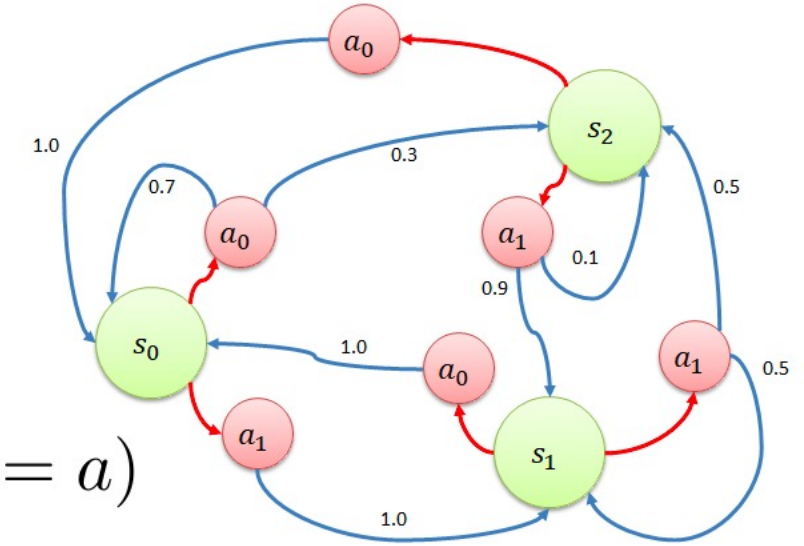
- **States are affected by previous actions**
- **Discrete actions**
- **Agent takes action A_t based on observed state and receives feedback as rewards R_t**
- $M = < S, A, T, R, \gamma >$
  - **States: encode world configurations**
  - **Actions: choices made by the agent**



Markov Decision Process

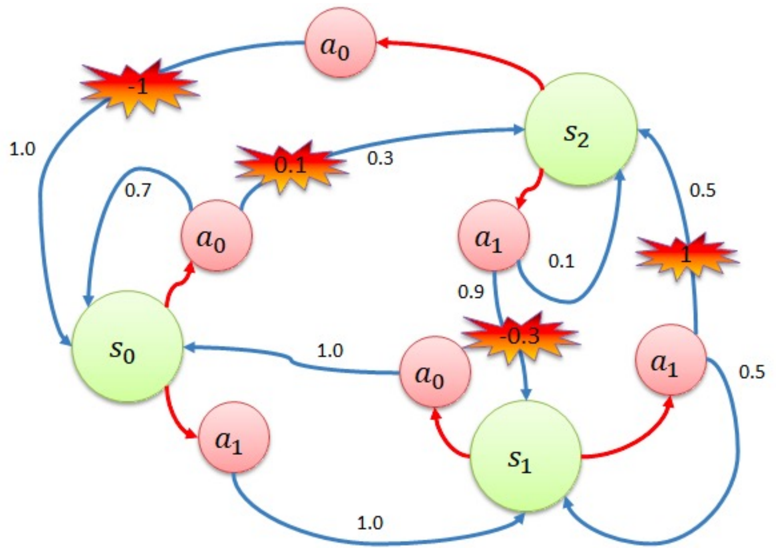
- **States are affected by previous actions**
- **Discrete actions**
- **Agent takes action A_t based on observed state and receives feedback as rewards R_t**
- $M = < S, A, T, R, \gamma >$
    - **Transition function: how the world evolves under actions**
    - **Stochastic!**

$$T(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$$



Markov Decision Process

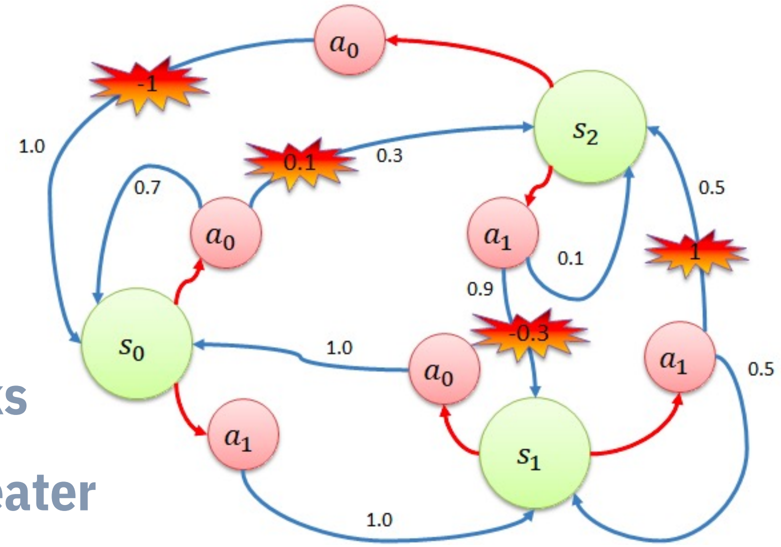
- **States are affected by previous actions**
- **Discrete actions**
- **Agent takes action A_t based on observed state and receives feedback as rewards R_t**
- $M = < S, A, T, R, \gamma >$
  - **Rewards: feedback signal to agent**



$$R(s, a) = E[r_t | s_t = s, a_t = a]$$

Markov Decision Process

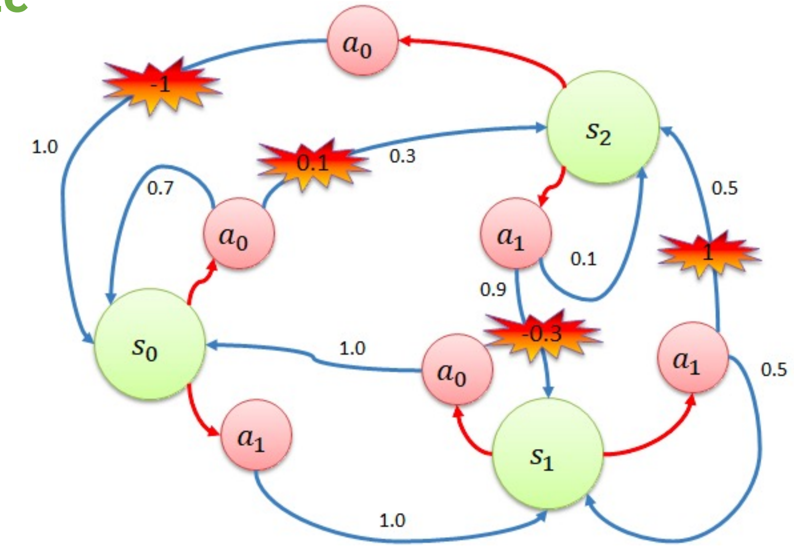
- **States are affected by previous actions**
- **Discrete actions**
- **Agent takes action A_t based on observed state and receives feedback as rewards R_t**
- $M = < S, A, T, R, \gamma >$
  - $\gamma \in [0, 1)$ **discounting factor for future rewards**
  - **Particularly important for *continuing* tasks**
  - **The value of something now is usually greater than in the future**



Outline

Markov Decision Process

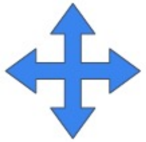
- **States are affected by previous actions**
- **Discrete actions**
- **Agent takes action A_t based on observed state and receives feedback as rewards R_t**
- $M = < S, A, T, R, \gamma >$
- **Markov**:
  - **"Future is independent of the past, given the present"**
  - **Fully observable environments**



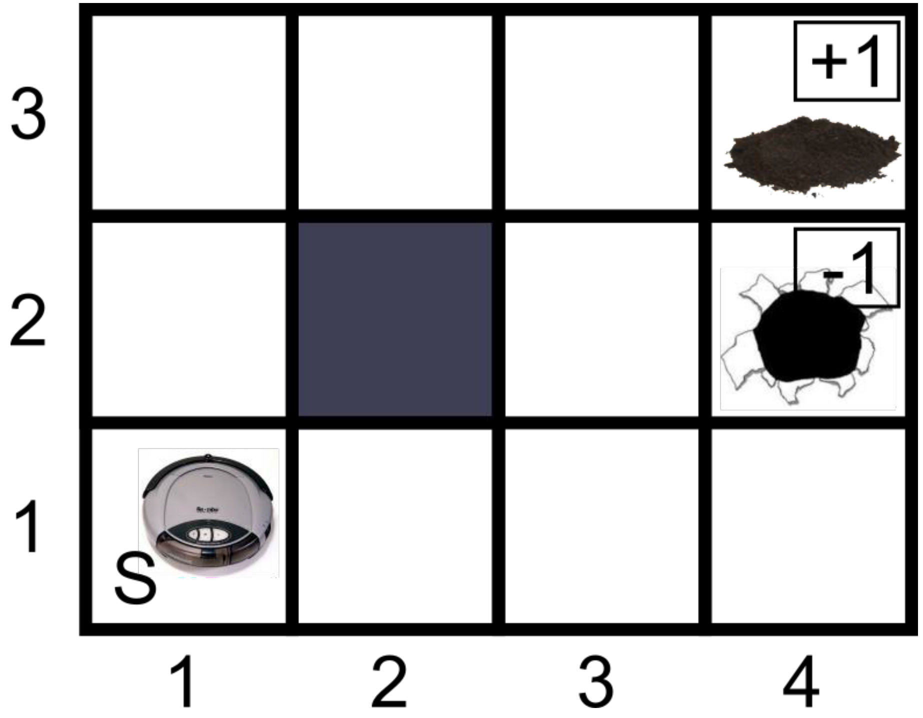
$$P(s_{t+1}|s_t) = P(s_{t+1}|s_0, \ldots, s_t)$$

An example

- **Cleaning robot**
- **Actions:**



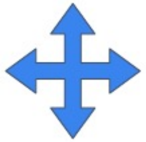
- **Reward:**
  - **+1 for finding dirt**
  - **-1 for falling into hole**
  - **-0.001 for every move**



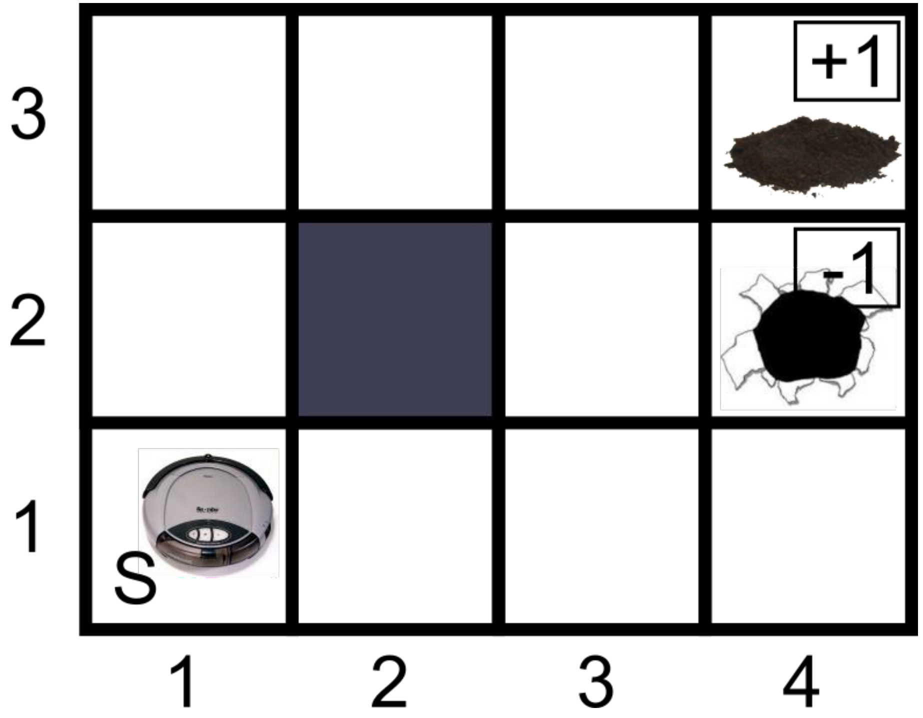
- **Episodic task: agent has repeated episodes of interaction (e.g., many attempts at cleaning the room)**
- **Goal: maximize total reward**

An example

- **States:**
  - **Position on grid e.g.,**
    - **S=(1,1), goal=(4,3)**
- **Actions:**



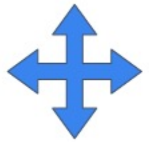
- **Reward:**
  - **+1 for finding dirt**
  - **-1 for falling into hole**
  - **-0.001 for every move**



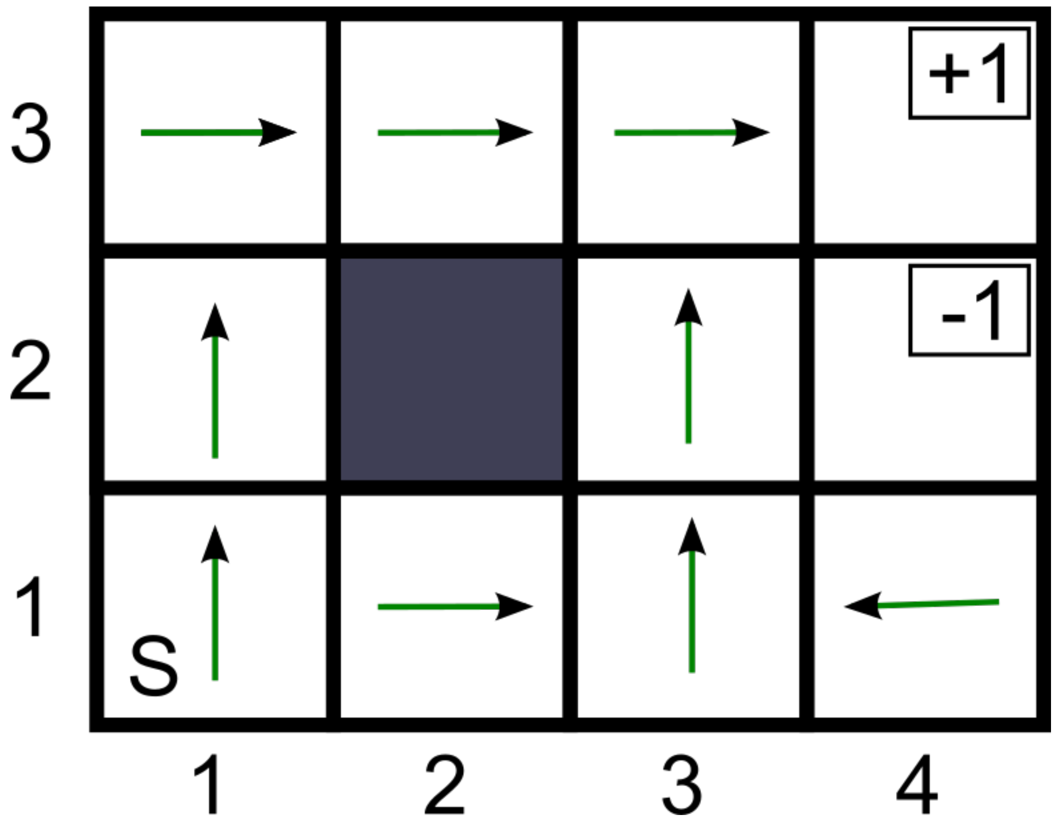
- **Episodic task: agent has repeated episodes of interaction (e.g., many attempts at cleaning the room)**
- **Goal: maximize total reward**

What is the optimal policy?

- **States:**
  - **Position on grid e.g.,**
    - **S=(1,1), goal=(4,3)**
- **Actions:**



- **Reward:**
  - **+1 for finding dirt**
  - **-1 for falling into hole**
  - **-0.001 for every move**



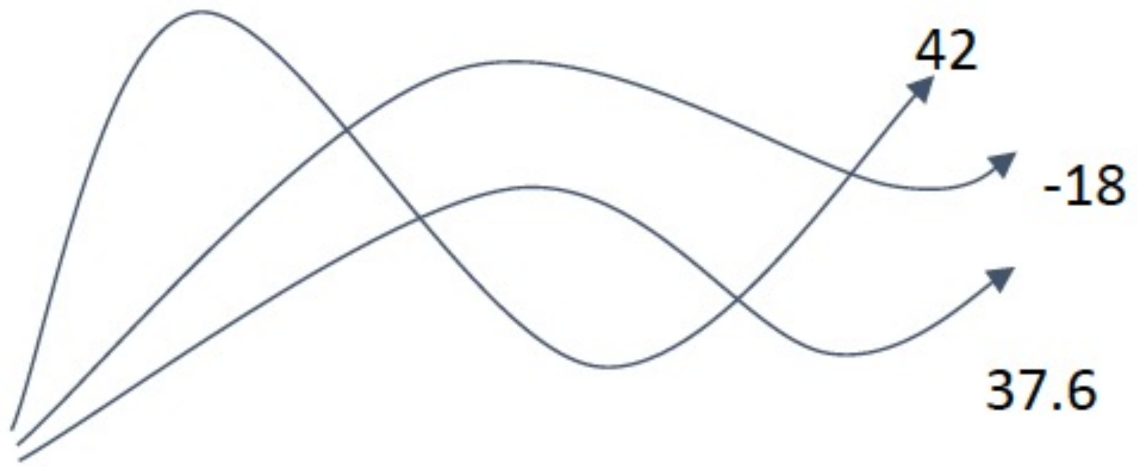
- **Episodic task: agent has repeated episodes of interaction (e.g., many attempts at cleaning the room)**
- **Goal: maximize total reward**

Outline

Evaluating behaviors

- **Many different trajectories are possible through a space**
- **Use the total *discounted accumulated rewards* to evaluate them (also called *return*)**

$$G_t = \sum_{k=t+1}^{N} \lambda^{k-t-1} R_k$$

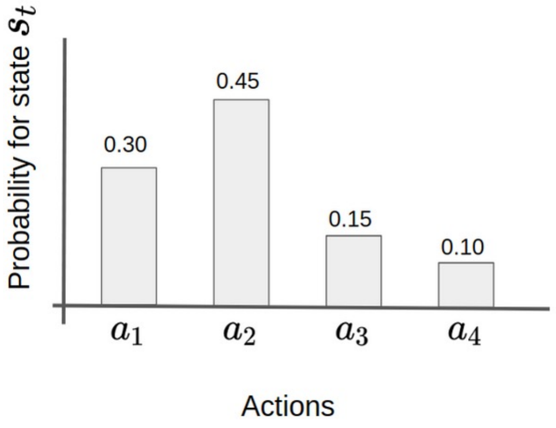


Rewards

- **Scalar feedback signal**
- **Encode (un)desirable features of behaviors:**
  - **Winning/losing, collisions, taking expensive actions, recommending incorrect medication, ...**
- **Typically:**
  - **Sparse**
  - **Delayed**
- **Credit assignment problem**

Policies

- **A *policy* (or behavior or strategy) π is any mapping from states to actions to take**
  - **Deterministic $\pi(s)$**
  - **Stochastic $\pi(a|s) = P(a_t = a|s_t = s)$**



- **Optimal policy π^***
  - **Accumulates maximal rewards over a trajectory**
  - **This is what we want to learn!**

Immediate vs delayed rewards

- **Cannot only rely on the *immediate* reward received**
- **Tradeoff: don't just maximize short term gain**

- **Notion of *value* to encode the goodness of a state, considering a policy running into the future**
  - **Represented as a *value function***

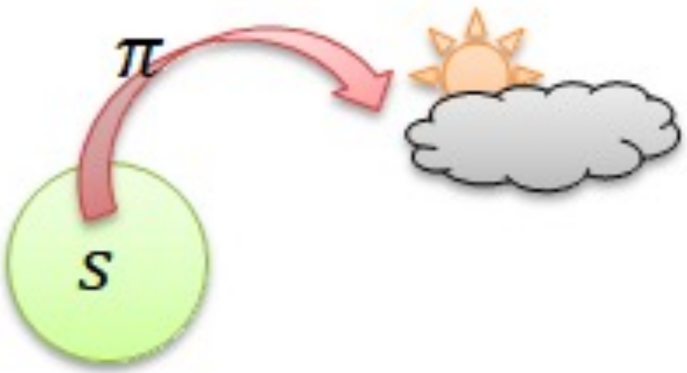
Value methods revisited

Value function:

The expected return (G_t) starting at state s and then executing policy π

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\sum_{t=0}^{\infty} \gamma^t r_{\pi(s_t)}(s_t, s_{t+1})\}$$

"How good is s under π?"



Outline

Value methods: Recursion

$V(s) \implies$ expected return starting at s and following π

Suggests dependence on value of next state s'

Bellman Equation:

$$\underbrace{\boxed{V^\pi(s)}}_{\text{value of } s} = \underbrace{\boxed{R(s,\pi(s))}}_{\text{immediate reward}} + \gamma \underbrace{\boxed{\sum_{s'}}}_{\text{for all possible next states}} \underbrace{\boxed{T(s,\pi(s),s')}}_{\text{the probability of reaching that state with } \pi} \underbrace{\boxed{V^\pi(s')}}_{\text{value of } s'}$$

$$V^\pi(s) = E_\pi\{R_t | s_t = s\} = E_\pi\{\sum_{t=0}^{\infty} \gamma^t r_{\pi(s_t)}(s_t, s_{t+1})\}$$

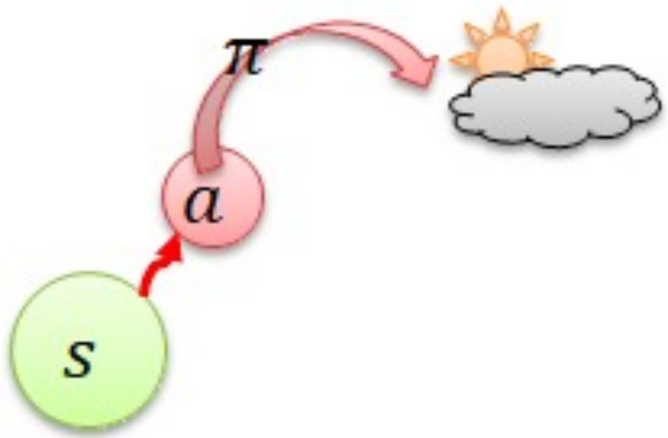
Outline

Action-value methods

$$\boxed{Q(s,a)} = \sum_{s'} \underbrace{\boxed{T(s,a,s')}}_{\text{transition probability}} \left(R(s,a,s') + \gamma \boxed{V(s')}\right)$$

The expected return (G_t) starting at state s and executing action a, and then following policy π

”How good is a in s under π?”



Optimal policies and value functions

$$\pi^*(a|s) = \begin{cases} 1 \; if \; a = argmax \; Q^*(s,a) \\ 0 \qquad\qquad otherwise \end{cases}$$

Move in direction of greatest value

Finding Q^*(or V^*) is equivalent to finding π^*

Optimal policies are greedy w.r.t. to Q^*(or V^*)

Every MDP has at least one optimal policy

Solving Bellman

Given the Bellman equation

$$V^*(s) = \max_a \{R(s,a) + \gamma \sum_{s'} T(s'|s,a) V^*(s')\}$$

Solve this as a large system of value function equations

- **But: non-linear (max operator)**
- **So: solve iteratively**

The goal (reminder):

- **Learn how good each state of the world is, when looking perfectly into the future**

Outline

Dynamic Programming

Value Iteration: Dynamic Programming

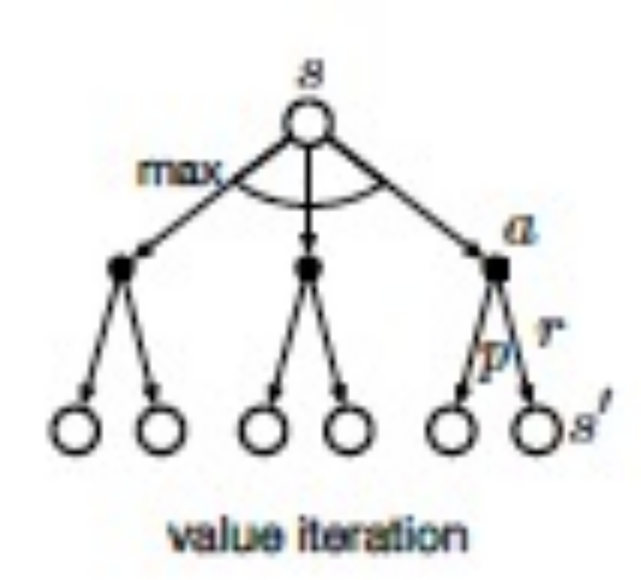
Iteratively update V

Value iteration

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$



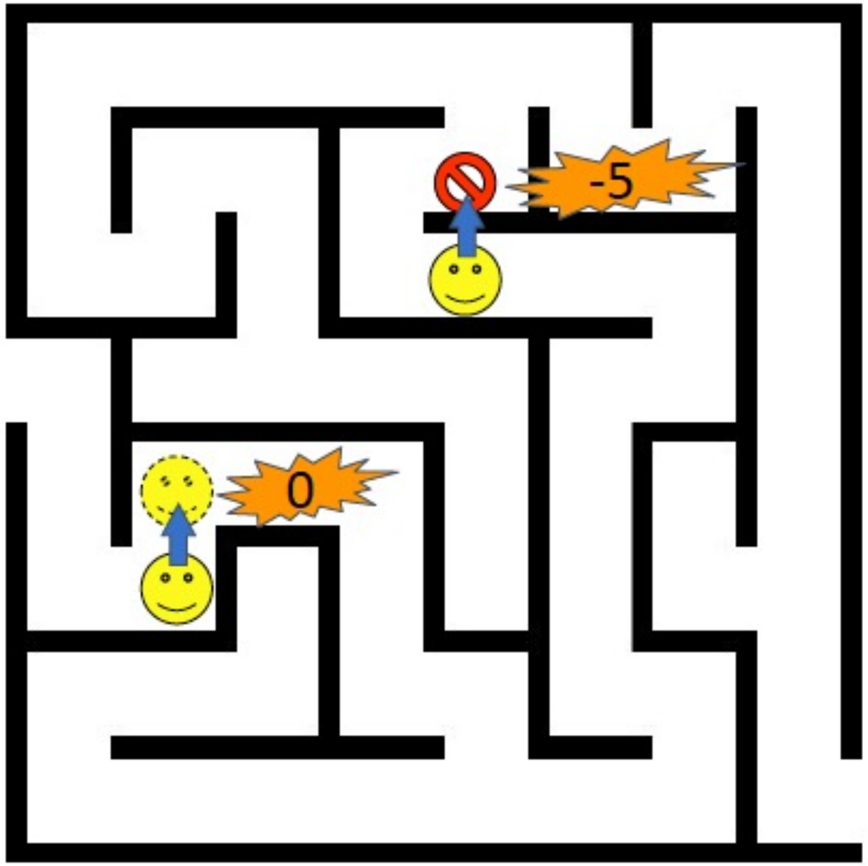
But it requires the full MDP! $M = < S, A, T, R, \gamma >$

In general, T and R are unknown. Also, S can be very large!

Data Generation from environment

T and R are unknown!

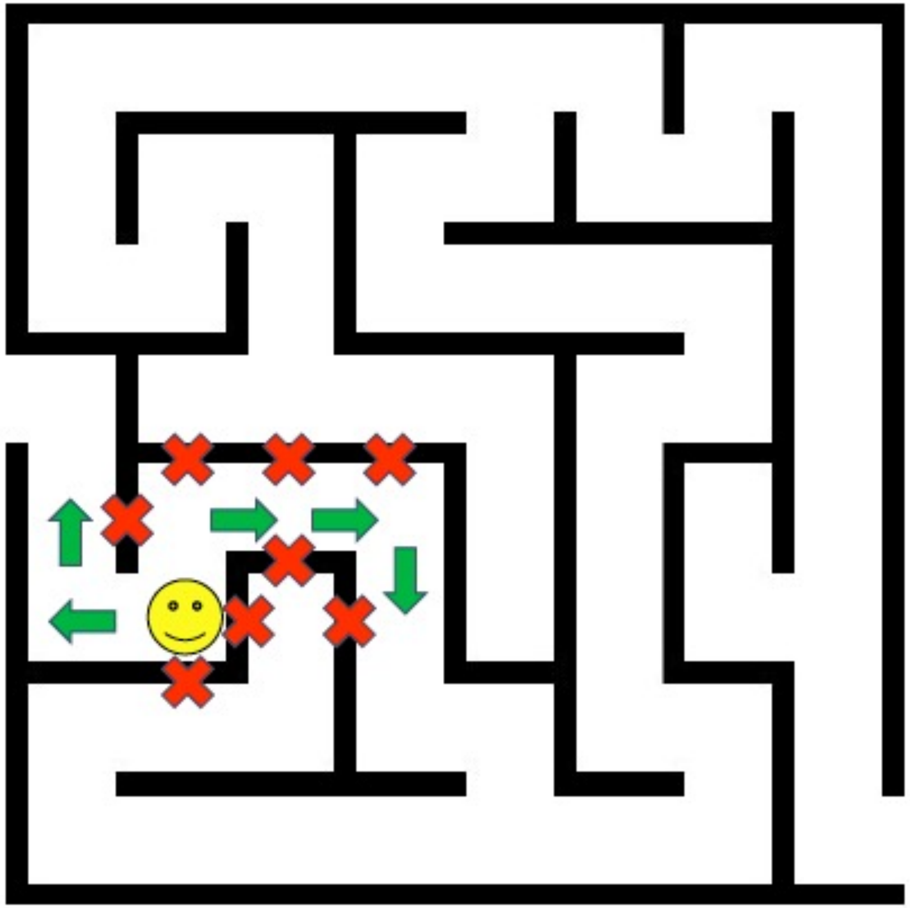
Instead, generate samples of training data (s, a, r, s') from the environment



Learning from Experience

We need:

- **An action selection strategy (e.g., ϵ-greedy)**
- **Some model to keep track of and learn value function**



TD Learning

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha[\underbrace{R + \gamma V(S')}_{\text{Target estimate of return}} - V(S)]$
    $S \leftarrow S'$
  until S is terminal

$$NewEstimate \leftarrow OldEstimate + StepSize \underbrace{\left[Target - OldEstimate\right]}_{\text{TD error}}$$

Notice how we learn update/learn as we are acting

Learning with actions

We can now learn by estimating $V(s)$ from experience

But:

- **Not learning about actions A**
- **We would rather learn $Q(s, a)$**
  - **For easier policy extraction!**
  - **$V(s)$ requires a one-step lookahead model**

SARSA

Learn from (s, a, r, s', a')

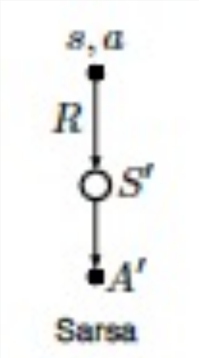
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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)
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

act

lookahed

learn



Converges with probability 1 to an optimal policy and action-value function as long as all state-action pairs are visited an infinite number of times

Outline

On and off policy

Where did we get the a'?

- **Taking the next action under Q**
- **This is an *on policy* algorithm**

What about *off policy*?

- **Learn about optimal policy while exploring**
- **Reuse experience from other policies**
- **Learn from observations**

Outline

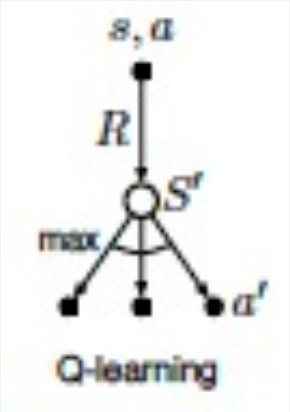
Q-Learning

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

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
  until S is terminal

act

learn



take the best next action
(so far)

Converges with probability 1 to an optimal policy under similar conditions but faster than SARSA

Outline

Application to Malaria Control

- **Malaria is a life-threatening disease**
  - **One of leading causes of morbidity and mortality in Africa (90% illness and 91% death cases, 2018)**
- **Need for Malaria control precautions (interventions)**
  - **Human decision makers (governments, NGOs, charities, etc.) use manual policies (interventions)**
  - **Complex policy spaces, inefficient for humans to explore with high returns**
  - **AI driven decision support system that will address inefficiencies**

Application to Malaria Control

[Submitted on 19 Jul 2021]

An Analysis of Reinforcement Learning for Malaria Control

Ndivhuwo Makondo, Arinze Lawrence Folarin, Simphiwe Nhlahla Zitha, Sekou Lionel Remy

Previous work on policy learning for Malaria control has often formulated the problem as an optimization problem assuming the objective function and the search space have a specific structure. The problem has been formulated as multi-armed bandits, contextual bandits and a Markov Decision Process in isolation. Furthermore, an emphasis is put on developing new algorithms specific to an instance of Malaria control, while ignoring a plethora of simpler and general algorithms in the literature. In this work, we formally study the formulation of Malaria control and present a comprehensive analysis of several formulations used in the literature. In addition, we implement and analyze several reinforcement learning algorithms in all formulations and compare them to black box optimization. In contrast to previous work, our results show that simple algorithms based on Upper Confidence Bounds are sufficient for learning good Malaria policies, and tend to outperform their more advanced counterparts on the malaria OpenAI Gym environment.

Comments:	27 pages including appendix and references. Submitted to the Journal of Artificial Intelligence Research (JAIR)
Subjects:	**Machine Learning (cs.LG)**; Artificial Intelligence (cs.AI)
Cite as:	arXiv:2107.08988 **[cs.LG]**
	(or arXiv:2107.08988v1 **[cs.LG]** for this version)

Submission history

From: Ndivhuwo Makondo [view email]
[v1] Mon, 19 Jul 2021 16:00:40 UTC (2,860 KB)

RL for Malaria Control

- **Environment: Epidemiological models**
  - **OpenMalaria: https://github.com/SwissTPH/openmalaria**
  - **EpidemiOptim (Covid): https://github.com/flowersteam/EpidemiOptim**
  - **Ushiriki: https://github.com/IBM/ushiriki-policy-engine-library**
- **Malaria Control as an optimization problem**

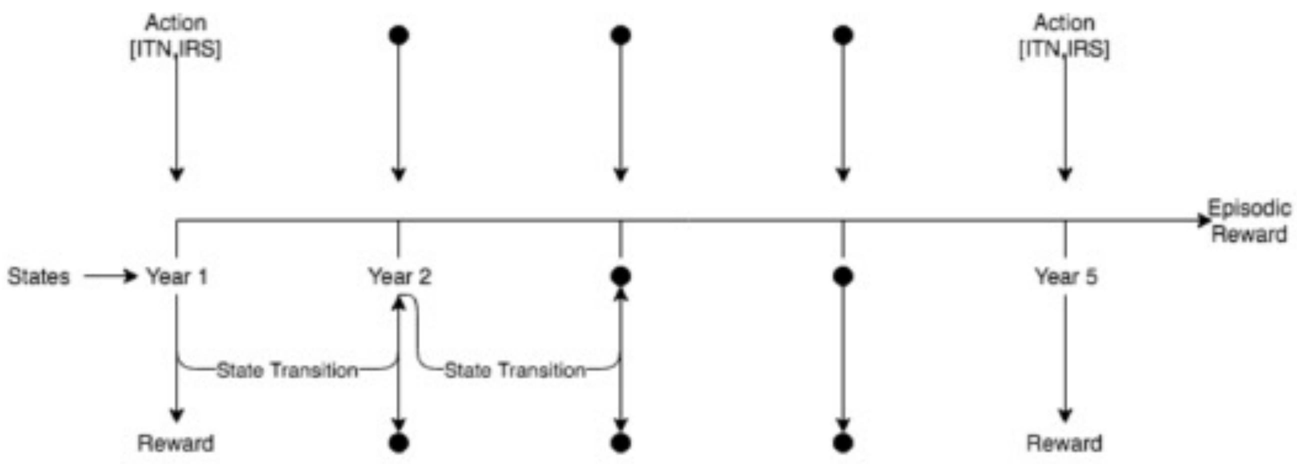
$$\max_{\mathbf{x} \in M} f(\mathbf{x})$$

where $\mathbf{x} \in M \subset \mathbb{R}^d$ **is a d-dimensional input space**

Let x_i **be the** i**th sample and** $r_i = f(x_i) + \epsilon_i$ **be noisy observation of** $f(x)$ **at** x_i **and noise value** ϵ_i

RL for Malaria Control

Outline



(Remy & Bent, 2020)

- **Sequential decision-making setting**
- **State: {1,2,3,4,5} years**
- **Action:** $\mathbf{a}_t \in A = \{a_{ITN}, a_{IRS}\}$
  - **where** $a_{ITN}, a_{ITN} \in [0,1]$
  - **ITN: long-lasting insecticide-treated nets**
  - **IRS: Indoor Residual Spraying**

RL for Malaria Control

- **Reward:**
  - **DALYs: Disability adjusted life years**
    - **Total years of life lost (YLL) due to fatality linked with malaria**
    - **Number of years of life with disability (YLD)**
  - **Simulated Costs (C_{int}):**
    - **Cost to treat and manage malaria episodes: healthcare system costs (HSC)**
    - **Cost to implement interventions which minimize malaria prevalence: intervention costs (IC)**
  - **Cost Effectiveness:**

$$C_{\mathrm{DA}} = \frac{\mathrm{HSC}_{\mathrm{int}} - \mathrm{HSC}_{\mathrm{noint}} + C_{\mathrm{int}}}{\mathrm{DA}}$$

RL for Malaria Control

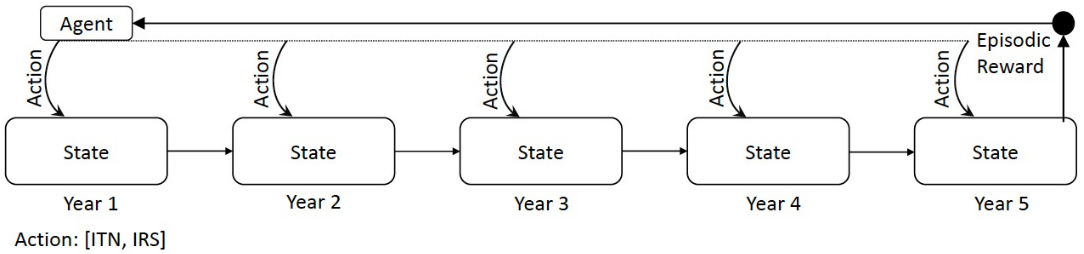
- **Policy:** $\pi_i(s) : S \to A$

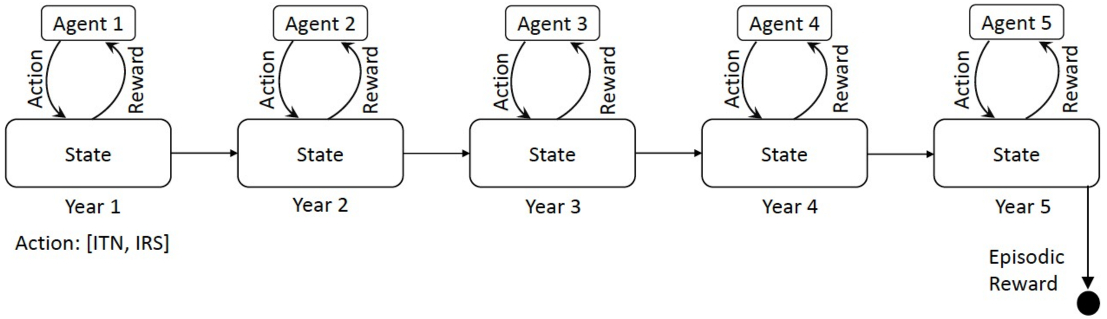
  $[(s_1, \mathbf{a}_1), \ldots, (s_5, \mathbf{a}_5)] \quad s_t \ (t \in \{1, \ldots, 5\})$

- **We seek optimal policy** $\pi_\star = \text{argmax}_{\pi \in \Pi} R(\pi)$
- **Interaction data:** $D_{1:i} = \{\pi_{1:i}, r_{1:i}\} \quad r_i = R(\pi_i)$
- **Solve as a general optimization problem:** $\max_{\mathbf{x} \in M} f(\mathbf{x})$
  - **Policy:** $\mathbf{x}_i = [a_{i,1}, a_{i,2}, \ldots, a_{5,1}, a_{5,2}]$
  - **No sequential structure. Cannot integrate state information**
  - **Bayesian optimization, Genetic Algorithm**
  - **Assumptions:** $f(x)$ **has no known structure such as linearity, concavity, and potentially not differentiable**

Which setting?

- **Multi-armed Bandit (Context-free)**
  - **A single state**
  - **Policy is a single action**
- **Contextual Bandit**
  - **Multiple states**
  - **States independent**
- **MDPs**
  - **Multiple correlated states**
  - **Actions impact states?**

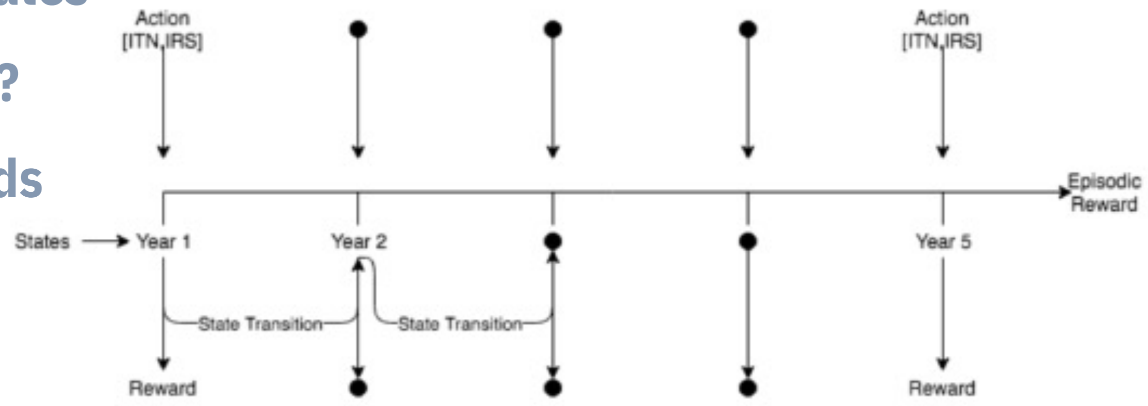




Outline

Which setting?

- **Multi-armed Bandit (Context-free)**
  - **A single state**
  - **Policy is a single action**
- **Contextual Bandit**
  - **Multiple states**
  - **States independent**
- **MDPs**
  - **Multiple correlated states**
  - **Actions impact states?**
  - **Actions impact rewards**



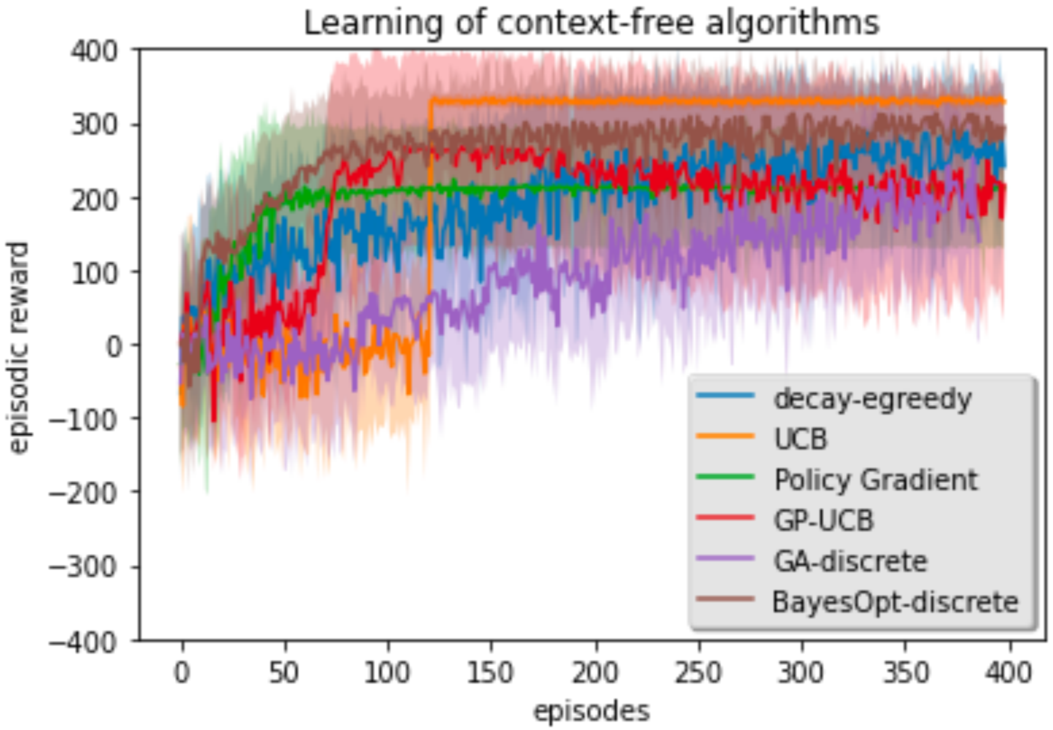
Algorithms

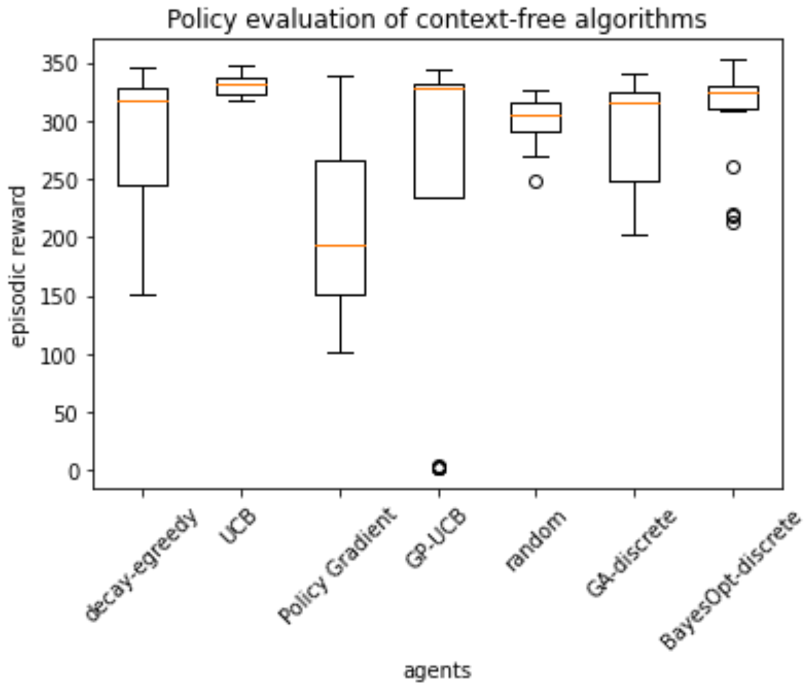
- **Multi-armed Bandit (Context-free)**
  - **Value-based methods: decay e-greedy, UCB, GP-UCB**
  - **Gradient Bandit with soft-max policy**
- **Contextual Bandit**
  - **Value-based methods: same with Q per state**
  - **Gradient Bandit with π per state**
- **MDPs**
  - **Value-based methods: Q-Learning with e-greedy and UCB**
  - **Policy Gradients: REINFORCE with neural network policy**
- **Baselines in multi-armed bandits and contextual bandits:**
  - **Random**
  - **Genetic Algorithm and Bayesian Optimization**

Outline

1. Speaker Bio
2. Previous Work
3. Intro to RL
4. Bandits
5. MDPs
6. **Malaria Control**
7. Summary

Results: context-free

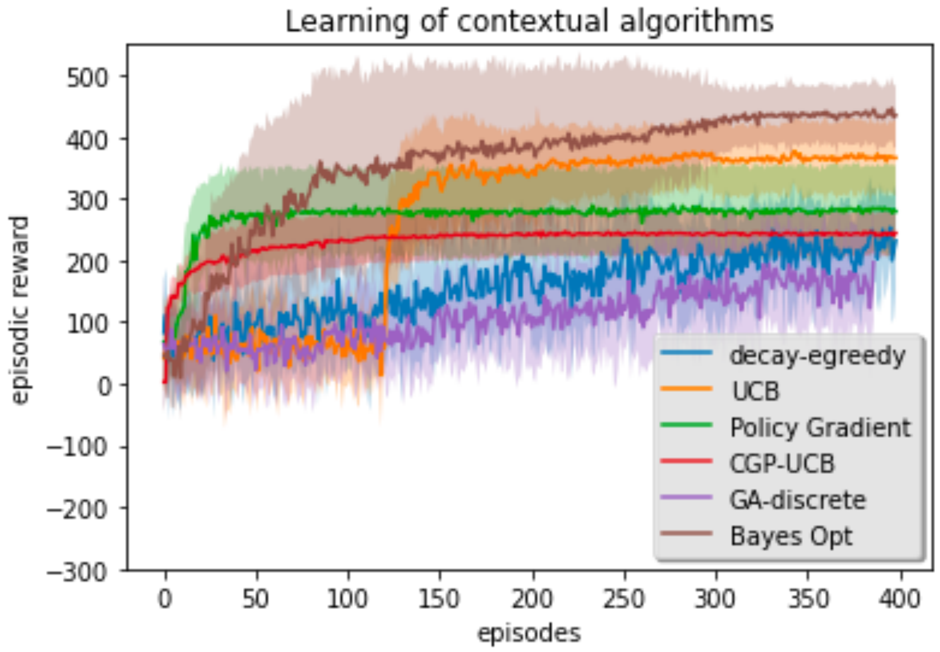


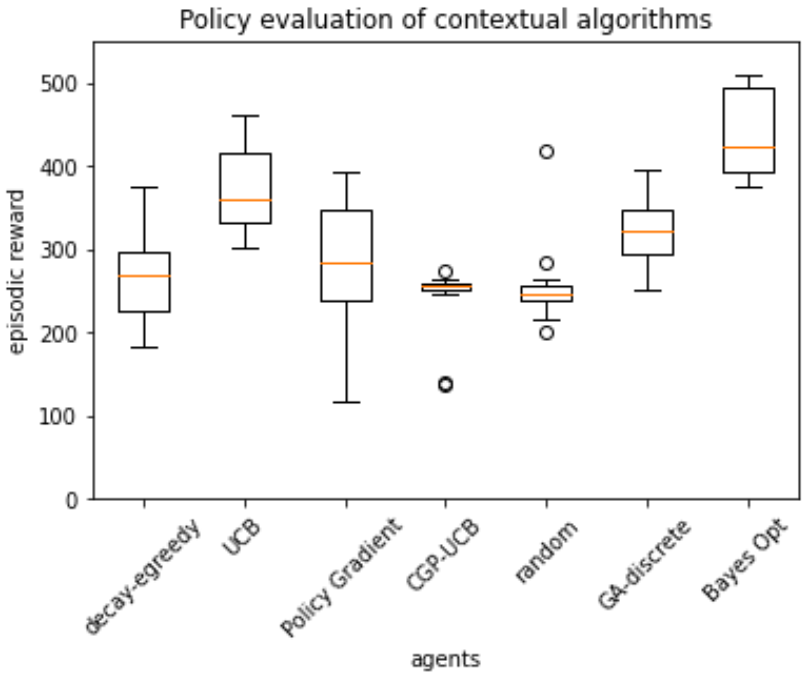


- **UCB performs better in the long run**
- **Only UCB significantly outperforms random baseline**
- **Policy gradients learn faster amongst RL**
- **Bayesian optimization is a better baseline**

Outline

Results: contextual

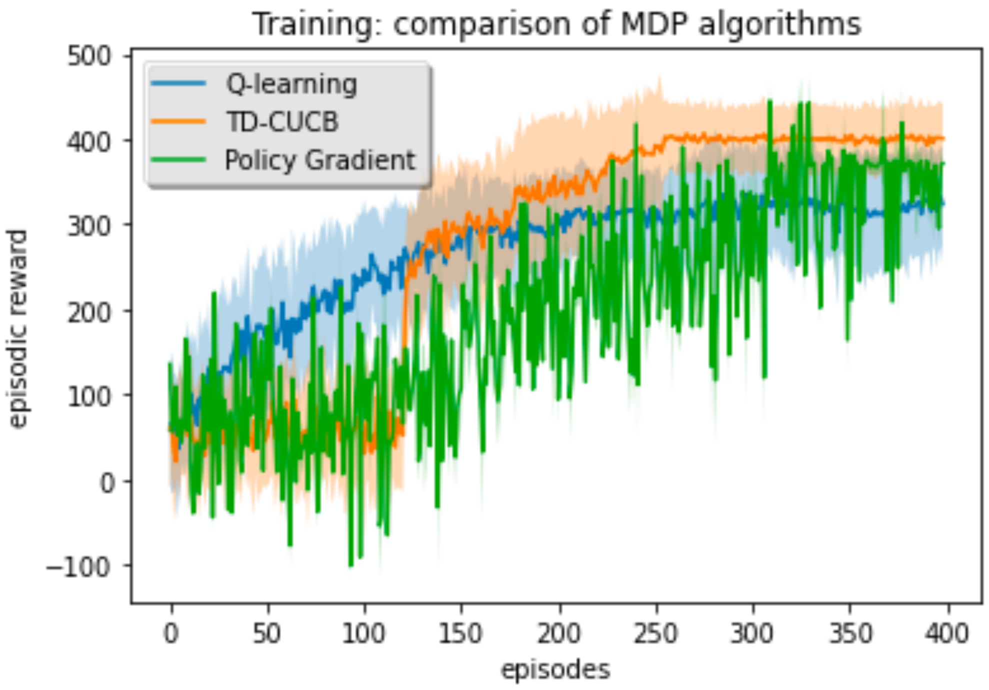


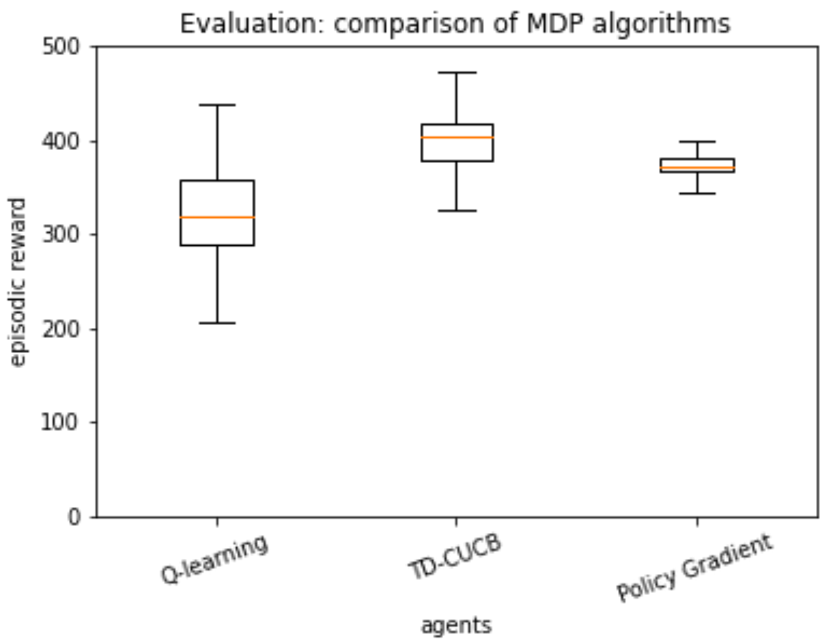


- **Learning is harder (random, e-greedy, policy gradient)**
- **UCB performs better than GP-UCB**
- **Bayesian optimization outperforms all**
- **Contextual bandit is a better setting (UCB, Bayes Opt, Policy gradients)**

Outline

Results: MDPs

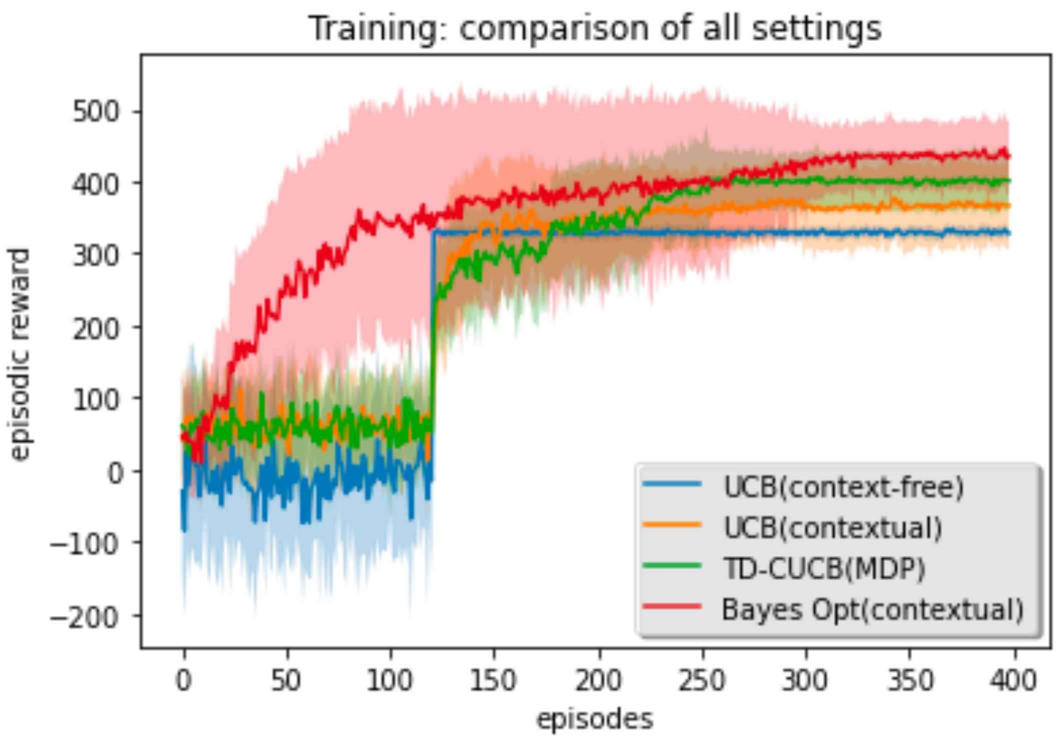


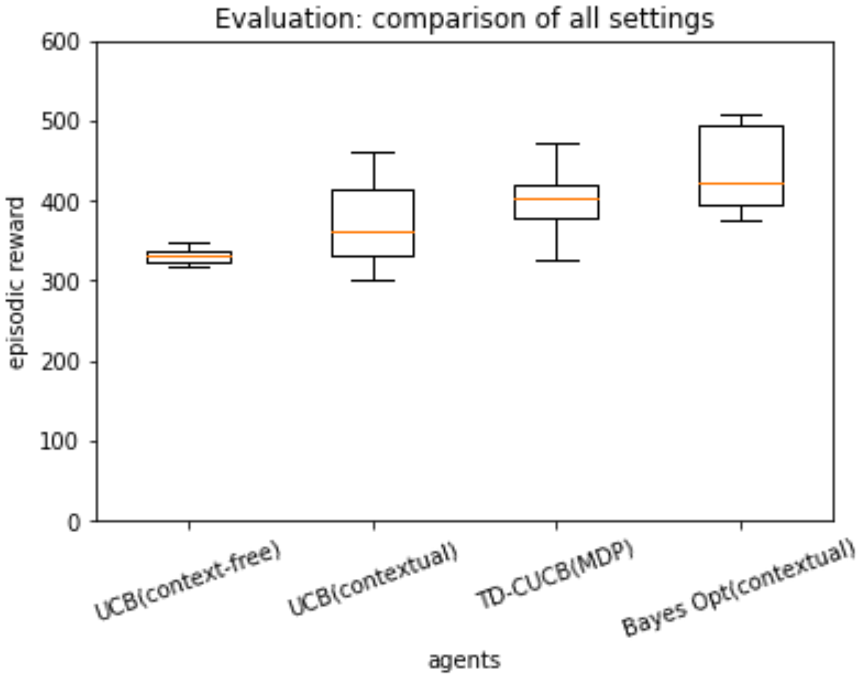


- **Q-Learning with UCB improves above e-greedy**
- **Policy gradients outperforms e-greedy**

Outline

Results: All settings





- **MDP setting better with Q-Learning and UCB**
- **Bayesian optimization is sample efficient!**
  - **Low dimensional search space (10 variables)**
  - **Could struggle with longer episodes (search space increases)**
  - **MDP with elements of Bayesian optimization?**

Summary

- **Need for learning and challenges**
- **Supervised learning vs Reinforcement learning**
- **The problem of learning behaviors**
  - **Model as Bandits or MDPs**
- **Components of solutions**
  - **Thinking about rewards**
  - **Policies, value functions**
  - **Exploration/exploitation**
- **Models are known**
  - **Value iteration: Dynamic Programming**
- **Learning from experience**
  - **TD, SARSA, Q-Learning**

Outline

Thank you