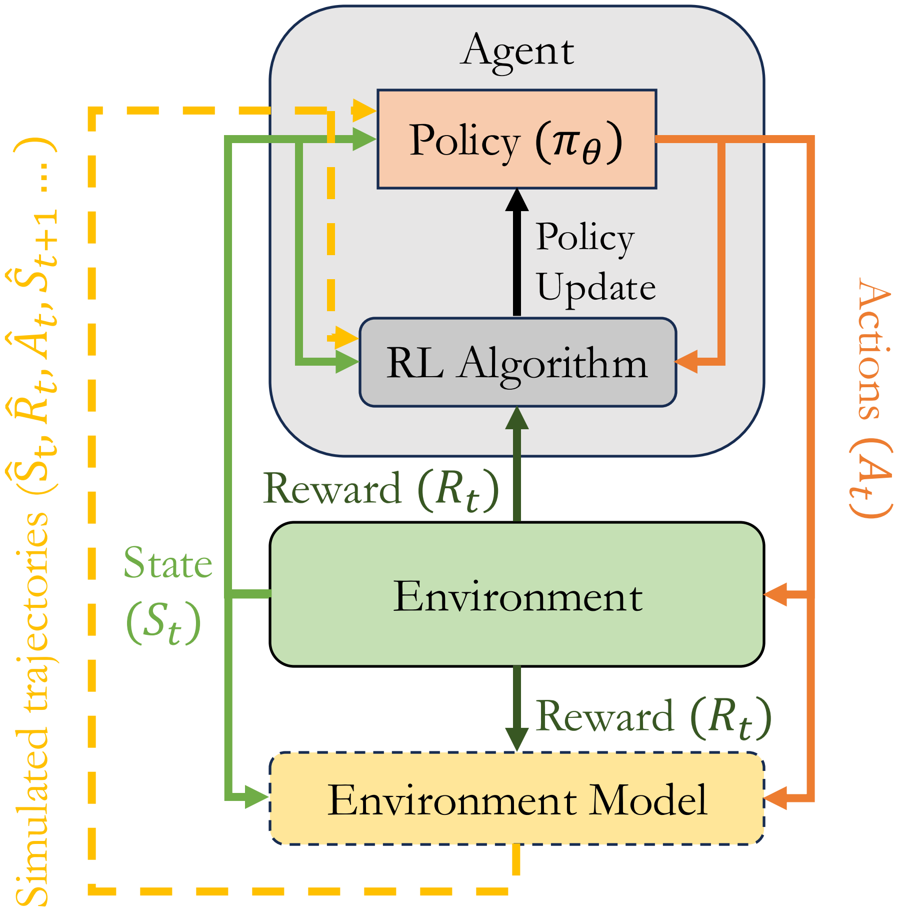
Reinforcement Learning

David I. Inouye

Credit: Souradip Pal (Spring 2024 GTA) drafted these slides.

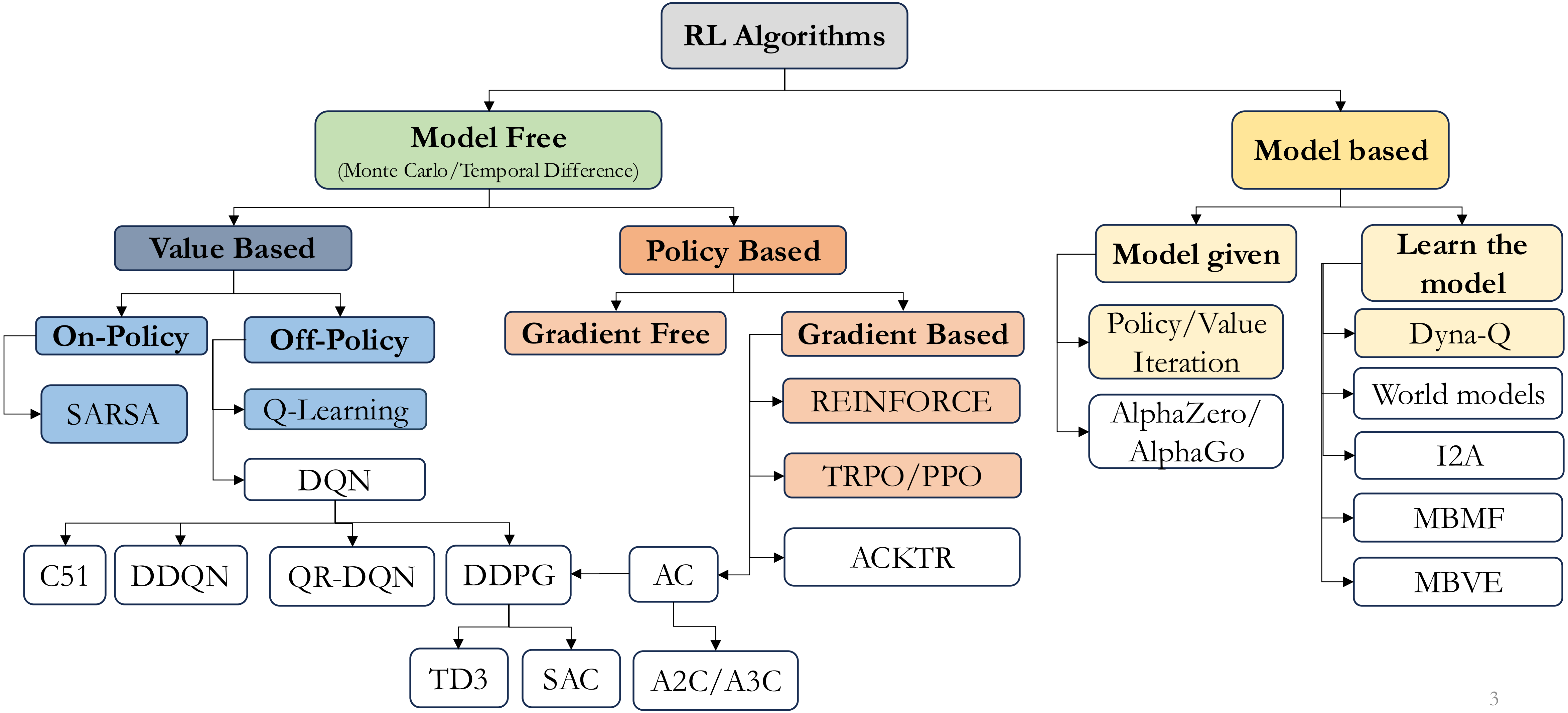
PURDUE UNIVERSITY
Elmore Family School of Electrical and Computer Engineering

Reinforcement Learning Algorithms Overview



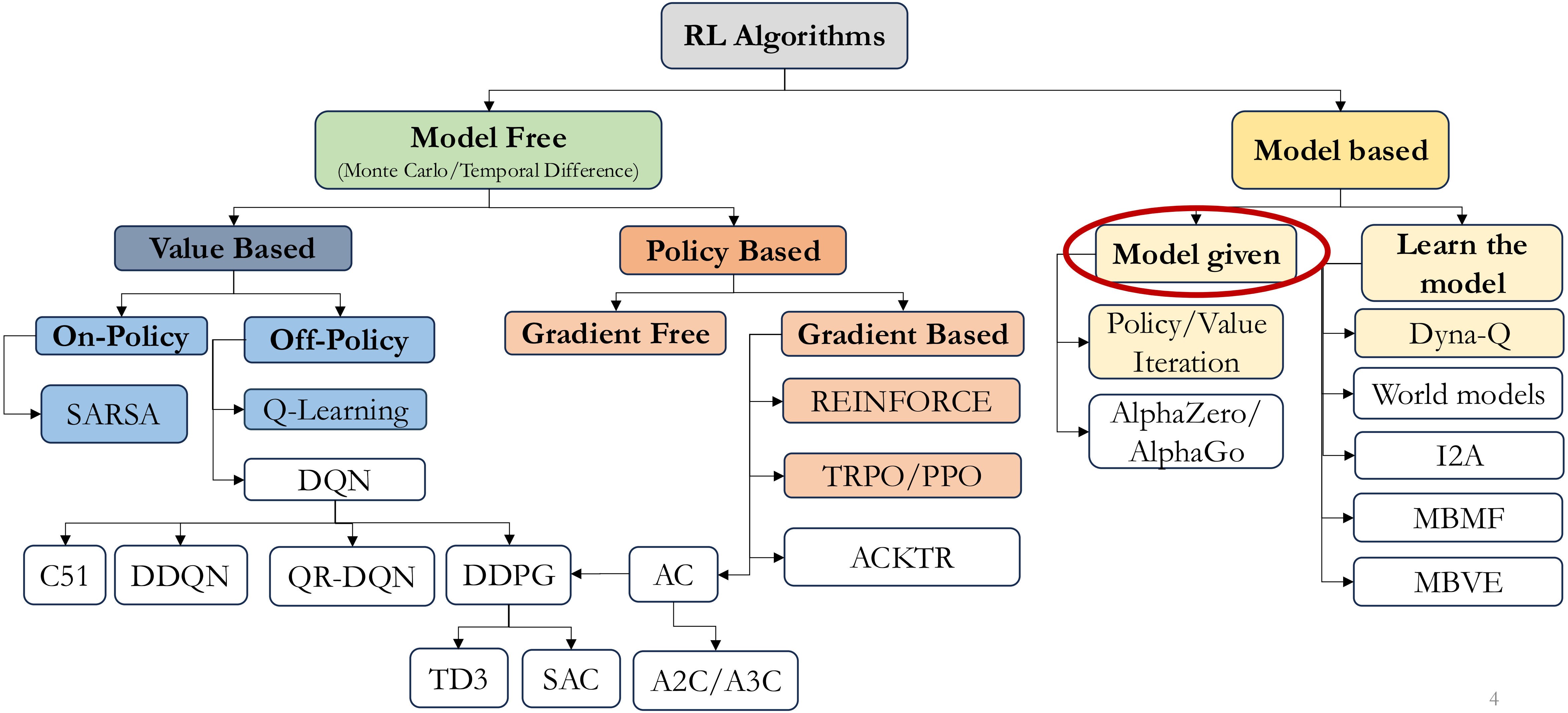
- Recall that our aim is to find the **optimal policy** which maximizes the **expected return** (discounted sum of future rewards)
- Policies can be compared based on value functions (policy ≈ value function), thus need a way to compute value function (**Prediction**) – **Policy Evaluation**
- Starting with an arbitrary policy improve the policy to reach optimal policy (**Control**) – **Policy Iteration**
  - Optimal policy can be constructed from optimal value function, improve value function - **Value Iteration**
- What if environment(MDP) is unknown?
  - **Estimate** value function via. reward sampling (**Model Free**)
  - Or learn a model of the environment (**Model Based**), then compute value function (simulated experience)
- What if MDP has continuous or infinite states?
  - Use **parameterized function approximators** for value function (**Value based**) or policy(**Policy Based**)
  - Search or learn parameters (**gradient free** or **gradient based** searching)

Categorizing RL Algorithms



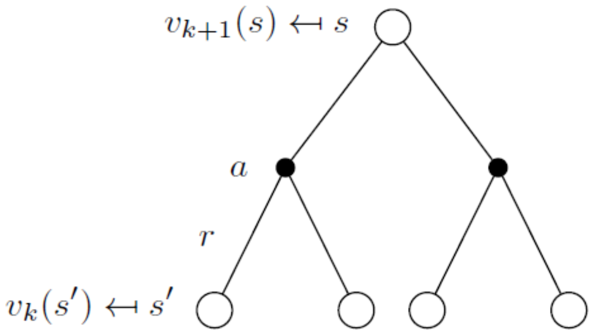
David I. Inouye, Purdue University

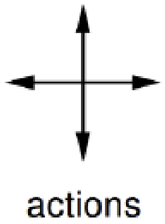
Categorizing RL Algorithms

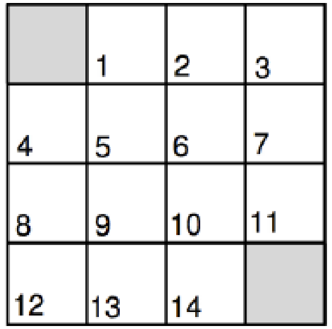


(1.A) Policy **Evaluation** – How good is your policy?

- Evaluate a given policy π, estimate v_π
- Also known as a **Prediction** problem
  - Input: Known MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
  - Output: Value function v_π
- Solution - Iterative application of **Bellman equation** and dynamic programming
  - At each iteration $k+1$, update $v_{k+1}(s)$ from $v_k(s')$ for all state s and successor states s'
  - $v_{k+1}(s) = \sum_{a,r,s'} \pi(a|s)\, p(s', r|s, a)[r + \gamma v_k(s')]$







Undiscounted
episodic MDP ($\gamma = 1$)

$r = -1$
on all transitions

Terminal state is gray

Random policy
$\pi(a|s) = 0.25$
$\forall\, s \in \mathcal{S}, a \in \mathcal{A}$

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

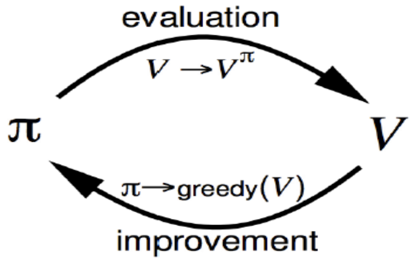
$k = 2$

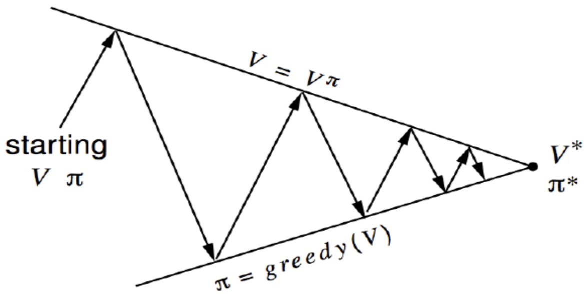
0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

Image Credit: Sutton and Barto book, Example 4.1, pp. 76.

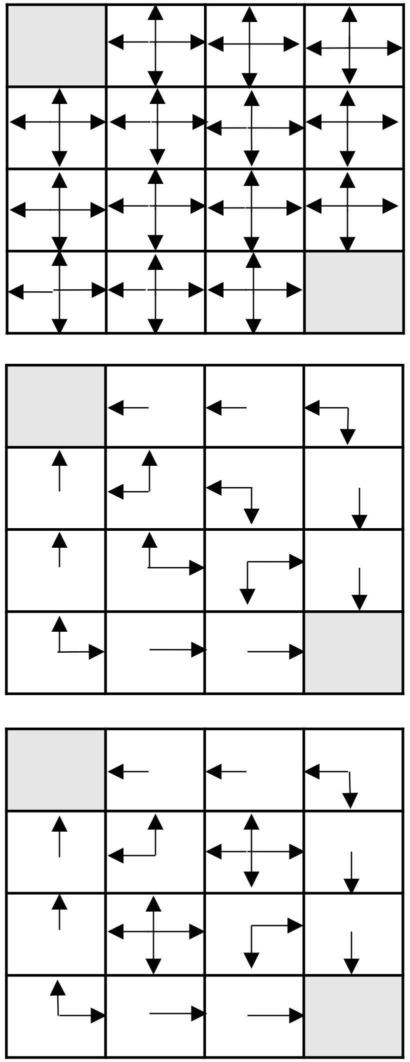
(1.B.1) Policy **Iteration** – How to improve a policy? How to find the optimal policy ?

- Given a policy π, find **optimal policy $\boldsymbol{\pi_*}$** (**<u>Control</u>**)
  - Evaluate the policy π, estimate v_π
  - Improve policy by acting **<u>greedily</u>** with respect to v_π
  - $\pi'(s) = \arg\max_a q_\pi(s,a) = \arg\max_a (r + \gamma \sum_{s'} p_{ss'}^a v_\pi(s'))$
  - $q_\pi(s, \pi'(s)) = \max_a q_\pi(s,a) \geq q_\pi\big(s, \pi(s)\big) = v_\pi(s)$
- If improvement stops, we have reached the optimal policy (also optimal value function)
  - $q_\pi(s, \pi'(s)) = \max_a q_\pi(s,a) = q_\pi(s, \pi(s)) = v_\pi(s)$
  - **Bellman Optimality equation** is satisfied
  - $v_\pi(s) = \max_a q_\pi(s,a) = \boldsymbol{v_*}(s)$ for all s





π_k v_k



$k = 1$, v_k:

0	-14	-20	-22
-14	-18	-20	-20
-20	-20	-18	-14
-22	-18	-14	0

$k = 2$, v_k:

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

$k = \infty$, v_k:

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

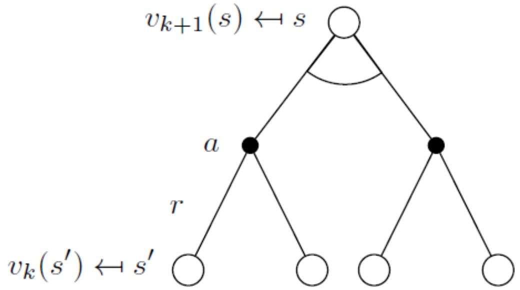
Image Credit: Sutton and Barto

(1.B.2) Value **Iteration** – Estimate optimal value function

- Find optimal value function v_* **directly** (get optimal policy π_* from v_*)
  - Unlike policy iteration, there is **<u>no explicit policy</u>**
  - Use **<u>Bellman Optimality equation</u>** to get $v_*(s)$ from the solution to subproblems $v_*(s')$
- Solution - Iterative application of Bellman optimality equation and dynamic programming
  - At each iteration $k+1$, update $v_{k+1}(s)$ from $v_k(s')$ for all state s and successor states s'
  - $v_{k+1}(s) = \max_a \sum_{s'} p(s', r|s, a)\,[r + \gamma v_k(s')]$

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \boldsymbol{\mathcal{R}}^{\mathbf{a}} + \gamma \boldsymbol{\mathcal{P}}^{\mathbf{a}} \mathbf{v}_k$$



v_k

$k = 1$

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$k = 2$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = \infty$

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

v_*

π_*

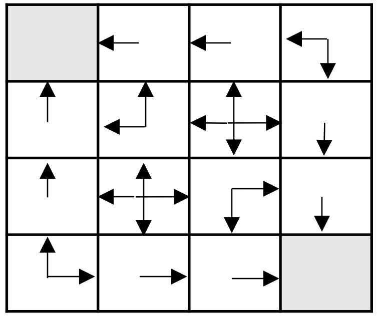
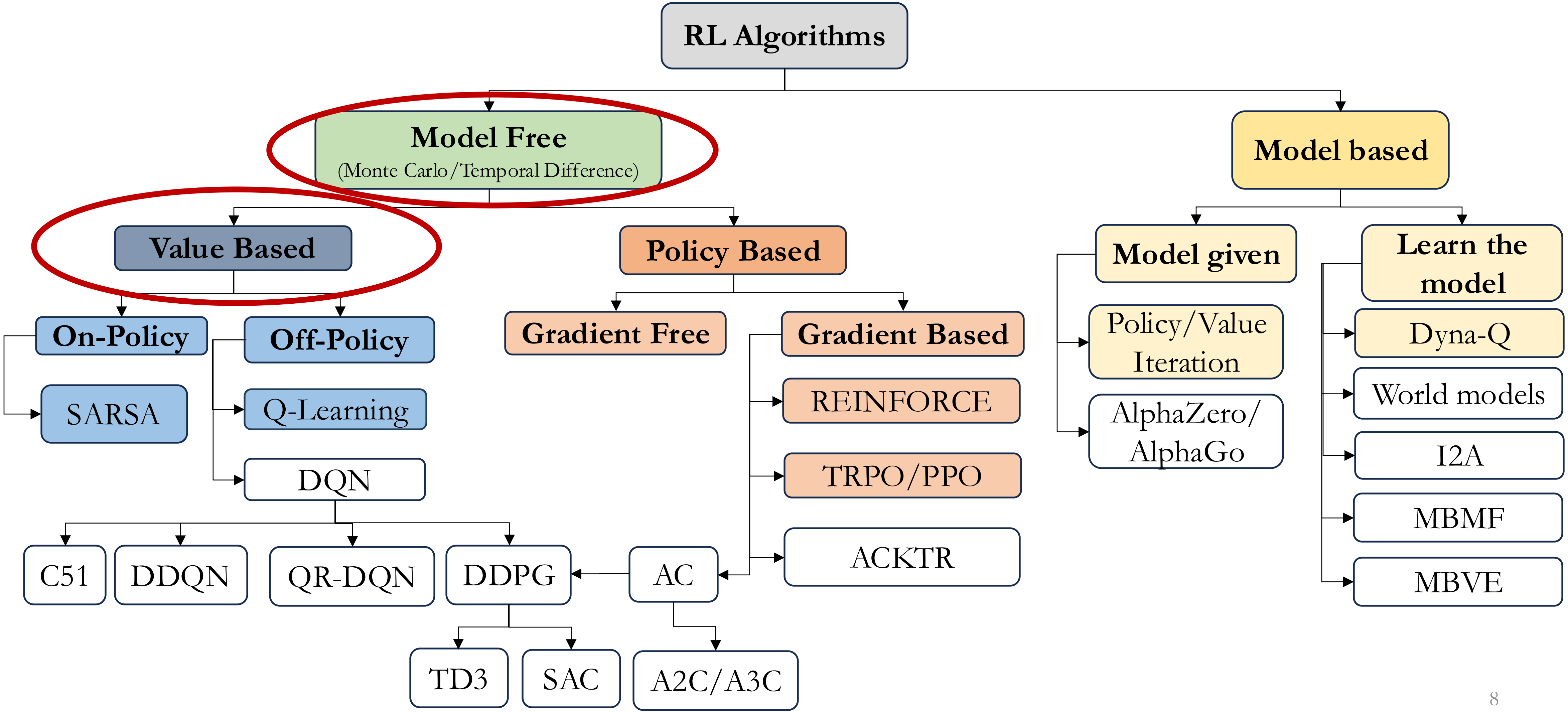


Image Credit: Sutton and Barto

Categorizing RL Algorithms



(2.A.1) Monte Carlo Policy **Evaluation** - Estimate value function for unknown MDPs (Model Free Prediction)

- No knowledge of MDP transitions or rewards
  - Observe the environment by sampling trajectories
  - Learn directly from experience (multiple episodes)
- Estimate value function
  - Take the mean of the returns observed
  - Consider complete episodes
- Assumptions
  - Applicable to episodic MDPs
  - All episodes must terminate (**finite horizon MDPs**)

First(Every) -Visit MC Evaluation

- Initialize $N(s) = 0, G(s) = 0 \; \forall s \in \mathcal{S}$
- Loop
  - Sample episode following policy π $(S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T)$
  - For each state s
    - Define $G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots \gamma^{T-1} R_T$ as return from time step $\boldsymbol{t}$ onwards where $\boldsymbol{t}$ is the **first(every) time** the state $\boldsymbol{s}$ is visited until $\boldsymbol{T}$ (the end of the episode)
    - Increment counter of total first(every) visits $N(s) = N(s) + 1$
    - Increment total return $G(s) = G(s) + G_t$
    - Update estimate $\hat{v}_\pi(s) = G(s)/N(s)$

David I. Inouye, Purdue University

(2.A.2) Monte Carlo Policy **Evaluation** - Estimate value function for unknown MDPs (Model Free Prediction)

- MC updates can be done **incrementally**
  - Uses formula to calculate incremental mean μ_k of a sequence $x_1, x_2, \dots, x_k$
  - $\mu_k = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$
  - $\hat{v}_\pi(s) \leftarrow \hat{v}_\pi(s) + \frac{1}{N(s)}(G_t - \hat{v}_\pi(s))$
- Estimate **state-action value function** (q)
  - $\hat{q}_\pi(s,a) \leftarrow \hat{q}_\pi(s,a) + \frac{1}{N(s,a)}(G_t - \hat{q}_\pi(s,a))$
  - $\hat{q}_\pi(s,a) \leftarrow \hat{q}_\pi(s,a) + \alpha(G_t - \hat{q}_\pi(s,a))$, α can be viewed as **step size** or learning rate
- Limitations
  - High variance estimator, require lots of data
  - Episode must end before data from episode can be used to update

Every-Visit Incremental MC

- Initialize $N(s,a) = 0, G(s,a) = 0 \ \forall s \in \mathcal{S}, a \in \mathcal{A}$
- Loop
  - Sample episode following policy π $(S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T)$
  - For each state-action pairs (s, a)
    - Define $G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots \gamma^{T-1} R_T$ as return from time step $\boldsymbol{t}$ onwards where $\boldsymbol{t}$ is **every time** the state $\boldsymbol{s}$ is visited and action $\boldsymbol{a}$ is taken until $\boldsymbol{T}$ (the end of the episode)
    - Increment counter of total every visits $N(s,a) = N(s,a) + 1$
    - Update estimate $\hat{q}_\pi(s,a) = \hat{q}_\pi(s,a) + \frac{1}{N(s,a)}(G_t - \hat{q}_\pi(s,a))$

(2.B) Monte Carlo Policy **Optimization** - Estimate optimal value function for unknown MDPs (Model Free Control)

- No knowledge of MDP transitions or rewards
  - Observe the environment by sampling trajectories
  - Learn directly from experience (multiple episodes)
- Estimate the **optimal value function**
  - Use **Policy Iteration** approach
  - **MC** method in policy evaluation step
  - Greedy policy improvement on action-value function q
  - $\pi'(s) = \arg\max_a q(s,a)$
- Caveats
  - Greedy policy improvement on state value function (v) not possible, requires MDP model (i.e., only applicable to action-value function q)
  - Might not explore all states - Can be solved using **stochastic policy (ϵ-greedy)** to encourage continuous exploration

Deterministic Policy Improvement

- For each state $s \in \mathcal{S}$ (s in episode)
  - $\pi(s) = \arg\max_a \hat{q}(s,a)$

ϵ-Greedy Policy Improvement

- For each state $s \in \mathcal{S}$ (s in episode)
  - $a_* = \arg\max_a \hat{q}(s,a)$
  - $\pi(s,a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|}, if\ a = a_* \\ \frac{\epsilon}{|\mathcal{A}|}, otherwise \end{cases}$

(3.A) Temporal Difference(TD) **Learning** - Estimate value function for unknown MDPs (Model Free Prediction)

- Combination of **Monte Carlo** & **dynamic programming** methods
  - Immediately update estimate of v after each observed (s, a, r, s') tuple
  - TD learns from **incomplete episodes**, by bootstrapping
- Estimate value function
  - Update value toward estimated target return
  - TD target: $R_{t+1} + \gamma \hat{v}(S_{t+1})$
  - TD error : $\delta_t = [R_{t+1} + \gamma \hat{v}(S_{t+1})] - \hat{v}(S_t)$
- Advantages
  - Lower variance than MC (although biased estimator)
  - Can be used in episodic or infinite-horizon non-episodic **MDPs**

TD(0)/1-step TD Learning

- Initialize $\hat{v}_\pi(s) = 0 \ \forall s \in \mathcal{S}$, step size $\alpha \in (0, 1)$
- Loop
  - Sample state S_0
  - For each step t in episode until termination
    - Take action A_t based on policy π at S_t
    - Observe reward R_{t+1} & next state S_{t+1}
    - Update estimate $\hat{v}_\pi(S_t) \leftarrow \hat{v}_\pi(S_t) + \alpha([R_{t+1} + \gamma \hat{v}_\pi(S_{t+1})] - \hat{v}_\pi(S_t))$
    - $S_t \leftarrow S_{t+1}$

(3.B.1) Model-Free Control with TD Methods – SARSA (On-Policy TD Learning)

- Uses **TD** learning approach for policy evaluation
  - Estimate q of the policy π being followed
  - ϵ-Greedy policy improvement on action-value function q
- Estimate action value function
  - Update value toward estimated target return given $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ transition tuple (hence called **SARSA**)
  - SARSA target: $R_{t+1} + \gamma \hat{q}_\pi(S_{t+1}, A_{t+1})$
- Advantages
  - **On-policy** algorithm
  - Converges to the optimal action-value function. $\hat{q}_\pi(s,a) \to q_*(s,a)$

SARSA

- Initialize $\hat{q}(s,a)\ \forall s \in \mathcal{S}, a \in \mathcal{A}$ arbitrarily, $\hat{q}(s,a) = 0$ if s is terminal state, $\alpha \in (0,1)$
- Set initial ϵ-greedy policy π randomly
- Loop
  - Sample state S_0
  - Sample action A_0 at S_0 based on policy π
  - For each step t in episode
    - Take action A_t, observe R_{t+1} and S_{t+1}
    - Choose action A_{t+1} at S_{t+1} based on π
    - Update estimate $\hat{q}_\pi(S_t, A_t) \leftarrow \hat{q}_\pi(S_t, A_t) + \alpha([R_{t+1} + \gamma \hat{q}_\pi(S_{t+1}, A_{t+1})] - \hat{q}_\pi(S_t, A_t))$
    - Update policy $\pi(S_t)$ based on ϵ-greedy
    - $S_t \leftarrow S_{t+1}, A_t \leftarrow A_{t+1}$

On-policy versus Off-Policy Learning & Control

- On-policy learning
  - Learn to estimate and evaluate a policy π from experience obtained from following that policy (**<u>same policy for prediction and control</u>**)
  - Direct experience
- Off-policy learning
  - Learn to estimate and evaluate a policy π^t (called **<u>target policy</u>**) using experience gathered from following a different policy (called **<u>behavior policy</u>** π^b)
  - Indirect experience, learn from observing humans or other agents
  - Re-use experience generated from old policies
  - Learn about optimal policy while following exploratory policy
  - Learn about multiple policies while following one policy
- Need **importance sampling** corrections on returns along whole episode
  - $G_t^{\pi^t/\pi^b} = \left(\frac{\pi^t(A_t|S_t)}{\pi^b(A_t|S_t)} \frac{\pi^t(A_{t+1}|S_{t+1})}{\pi^b(A_{t+1}|S_{t+1})} \dots \frac{\pi^t(A_T|S_T)}{\pi^b(A_T|S_T)}\right) G_t$

David I. Inouye, Purdue University

(3.B.2) Model-Free Control with TD Methods – Q Learning (Off-Policy TD Learning)

- Q-learning is an **off-policy** RL algorithm on action-values q
- Maintain state-action q estimates for bootstrapping
  - Use the value of the best future action
  - Stochastic approximation like SARSA
- Estimate action value function
  - Next action is chosen using behavior policy $A_{t+1} \sim \pi_b(S_t)$
  - Consider all alternative successor action $A' \sim \pi(S_t)$, take best A' for update
  - Q-learning target: $R_{t+1} + \gamma \max_{A'} \hat{q}(S_{t+1}, A')$
- Advantages
  - No importance sampling required
  - Allows both behavior and target policies to improve

Q-Learning

- Initialize $\hat{q}(s,a)\ \forall s \in \mathcal{S}, a \in \mathcal{A}$ arbitrarily, $\hat{q}(s,a) = 0$ if s is terminal state, $\alpha \in (0,1)$
- Set initial ϵ-greedy policy π_b w.r.t $\hat{q}$
- Loop
  - Sample state S_0
  - Set ϵ-greedy policy π_b w.r.t $\hat{q}$
  - Sample action A_0 at S_0 based on policy π_b
  - For each step t in episode
    - Take action A_t, observe R_{t+1} and S_{t+1}
    - Update estimate $\hat{q}(S_{t+1}, A_{t+1}) \leftarrow \hat{q}(S_t, A_t) + \alpha([R_{t+1} + \gamma \max_{A'} \hat{q}(S_{t+1}, A')] - \hat{q}(S_t, A_t))$
    - Update policy π based on ϵ-greedy on $\hat{q}$
    - $S_t \leftarrow S_{t+1}$

(4.A) Value Function Approximation – Scaling up RL methods

- So far, we have been working with the tabular representation of the value functions $v(s)$ or $q(s,a)$ and policy $\pi(a|s)$ for finite and discrete MDPs
- But MDPs can be very large, need to **scale up** for large MDPs
  - Too many states and/or actions to store in memory, state space can be continuous
  - Too slow to learn the value of each state individually
- Solution – Estimate value function with **<u>function approximation</u>**
  - $\hat{v}(s,\boldsymbol{\theta}) \approx v_\pi(s)$ or $\hat{q}(s,a,\boldsymbol{\theta}) \approx q_\pi(s,a)$ where the value function is parameterized by $\boldsymbol{\theta}$
  - Update parameter $\boldsymbol{\theta}$ using MC and TD methods (supervised learning)
  - Generalizes to unseen states and/or actions
- Common Function Approximators (consider only differentiable ones)
  - Linear combination of features
  - Neural Networks
  - Nearest Neighbors
  - Decision Trees

(4.A.1) Linear Value Function Approx. by Gradient Descent

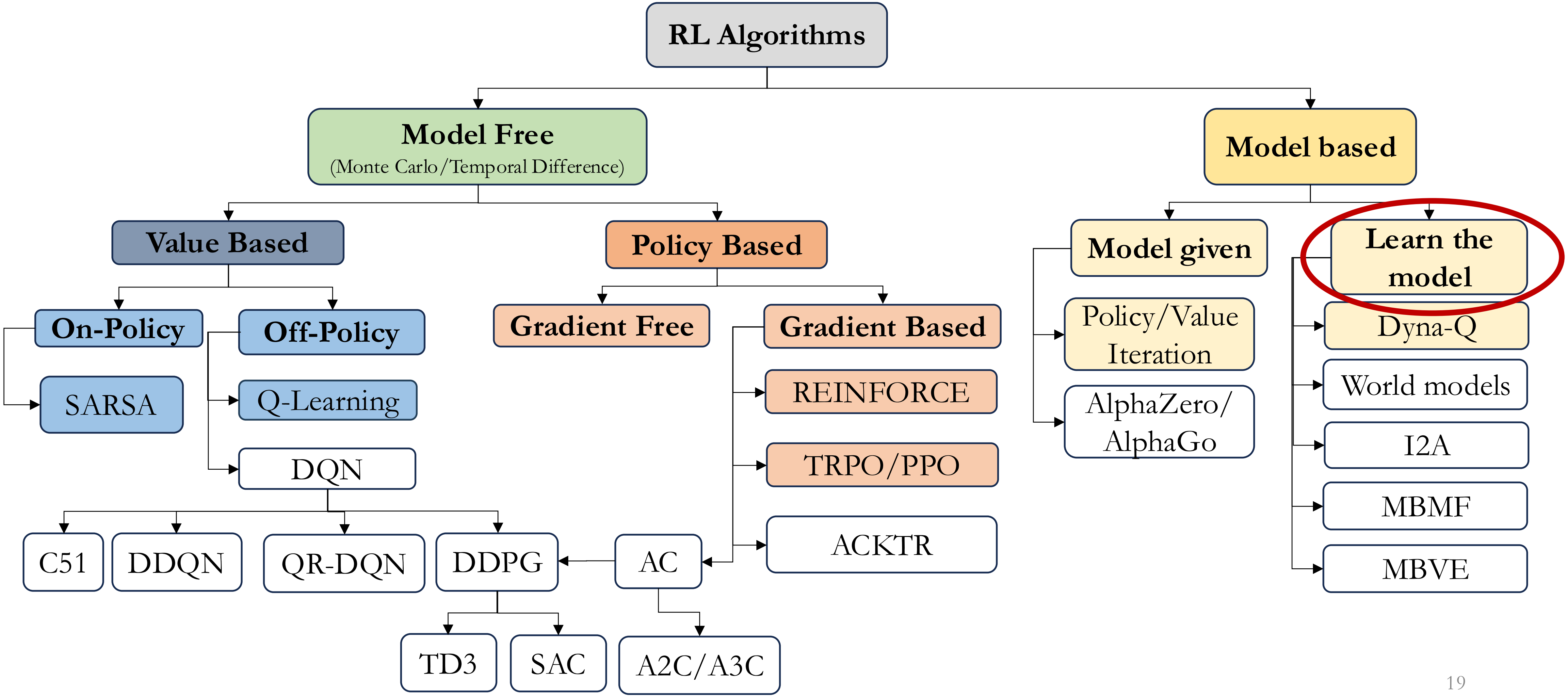
- Represent state by a feature vector $\mathbf{x}(s) = [x_1(s), x_2(s), \dots, x_n(s)]^T$
- Represent value function by a **linear combination of features**
  - $\hat{v}(s, \boldsymbol{\theta}) = \mathbf{x}(s)^T \boldsymbol{\theta}$, where $\boldsymbol{\theta} = [\theta_1, \theta_2, \dots, \theta_n]^T$
- Find parameter vector $\boldsymbol{\theta}$ minimizing the mean-squared error between approximate value function $\hat{v}(s, \boldsymbol{\theta})$ and true value function $v_\pi(s)$ (value objective function)
  - $J(\boldsymbol{\theta}) = \mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \boldsymbol{\theta})\right)^2\right]$
  - $J_{\text{linear}}(\boldsymbol{\theta}) = \mathbb{E}_\pi[(v_\pi(s) - \mathbf{x}(s)^T \boldsymbol{\theta})^2]$ (for linear value function approx.)
- Apply **gradient descent**(or SGD) to find local minimum by updating parameters
  - Update rule: $\Delta\boldsymbol{\theta} = -\frac{1}{2}\boldsymbol{\alpha}\nabla J(\boldsymbol{\theta}) = \boldsymbol{\alpha}\, \mathbb{E}_\pi\left[\left(v_\pi(s) - \hat{v}(s, \boldsymbol{\theta})\right)\nabla_{\boldsymbol{\theta}} \hat{v}(s, \boldsymbol{\theta})\right]$
  - SGD update rule: $\Delta\boldsymbol{\theta} = \boldsymbol{\alpha}\left[\left(v_\pi(s) - \hat{v}(s, \boldsymbol{\theta})\right)\nabla_{\boldsymbol{\theta}} \hat{v}(s, \boldsymbol{\theta})\right]$
  - SGD update rule for linear value function approx.: $\Delta\boldsymbol{\theta} = \boldsymbol{\alpha}\left[\left(v_\pi(s) - \hat{v}(s, \boldsymbol{\theta})\right)\mathbf{x}(s)\right]$
- Stochastic gradient descent converges to global optimum
- Seems great…but we don't know v_π!

David I. Inouye, Purdue University

(4.A.1) Incremental Prediction/Control Algorithm – MC/TD with Function Approx.

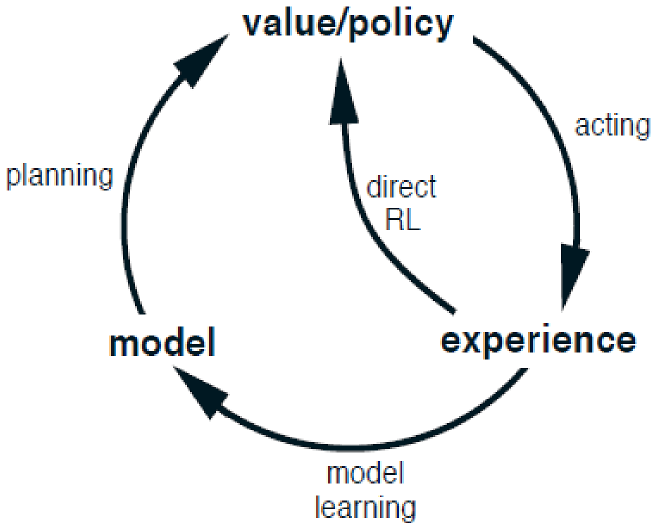
- In practice, we don't have true value function v_π for prediction, we only have rewards through environment interaction, thus substitute target for v_π
  - For MC, the target is the return G_t
    - $\Delta\boldsymbol{\theta} = \boldsymbol{\alpha}\left[\left(G_t - \hat{v}(S_t, \boldsymbol{\theta})\right)\nabla_{\boldsymbol{\theta}}\hat{v}(S_t, \boldsymbol{\theta})\right]$
  - For TD(0), the target is the TD target $R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{\theta})$
    - $\Delta\boldsymbol{\theta} = \boldsymbol{\alpha}\left[\left(R_{t+1} + \gamma\hat{v}(S_{t+1}, \boldsymbol{\theta}) - \hat{v}(S_t, \boldsymbol{\theta})\right)\nabla_{\boldsymbol{\theta}}\hat{v}(S_t, \boldsymbol{\theta})\right]$
- In control, approximate action-value function $\hat{q}(s, a, \boldsymbol{\theta})$, substitute target for true value of q_π
  - For MC, the target is the return G_t
    - $\Delta\boldsymbol{\theta} = \boldsymbol{\alpha}\left[\left(G_t - \hat{q}(S_t, A_t, \boldsymbol{\theta})\right)\nabla_{\boldsymbol{\theta}}\hat{q}(S_t, A_t, \boldsymbol{\theta})\right]$
  - For TD(0), the target is the TD target $R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{\theta})$
    - $\Delta\boldsymbol{\theta} = \boldsymbol{\alpha}\left[\left(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{\theta}) - \hat{q}(S_t, A_t, \boldsymbol{\theta})\right)\nabla_{\boldsymbol{\theta}}\hat{q}(S_t, A_t, \boldsymbol{\theta})\right]$
- (4.B) Approximate Policy Iteration - Do approximate policy evaluation using $\hat{q}(s, a, \boldsymbol{\theta}) \approx q_\pi$ followed by ϵ-greedy policy improvement

Categorizing RL Algorithms



Model-Based Reinforcement Learning – Integrating Learning and Planning

- Previous approach – Model Free RL
  - **No model** (unknown transition function $\mathcal{P}$ and reward function $\mathcal{R}$)
  - Learn value function/policy directly from experience
- New Approach – Model Based RL
  - First **learn(estimate) model** from experience
  - Plan for optimal value function/policy using learned model
  - Integrate learning and planning into a single architecture
  - Possible to efficiently learn model using supervised learning methods
  - Can understand model uncertainty
  - Model-based RL is only as good as the estimated model. When the model is inaccurate, planning process will compute a suboptimal policy.



Model $\mathcal{M}_\eta \xrightarrow{represents}$ MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

$\mathcal{M}_\eta = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ (η is the parameter)

$\mathcal{P}_\eta \approx \mathcal{P} \quad \mathcal{R}_\eta \approx \mathcal{R}$

Image Credit: Sutton and Barto

(5.A/B) Integrated Architectures – Dyna (Dyna-Q Algorithm)

- Dyna
  - Learn model from real experience
  - Learn and plan value function/policy from both real & simulated experience (Q-Learning)
- Involves one-step interaction(acting) with the environment and n steps planning
- Store experience, get better policy with fewer environment interactions

The Dyna Architecture

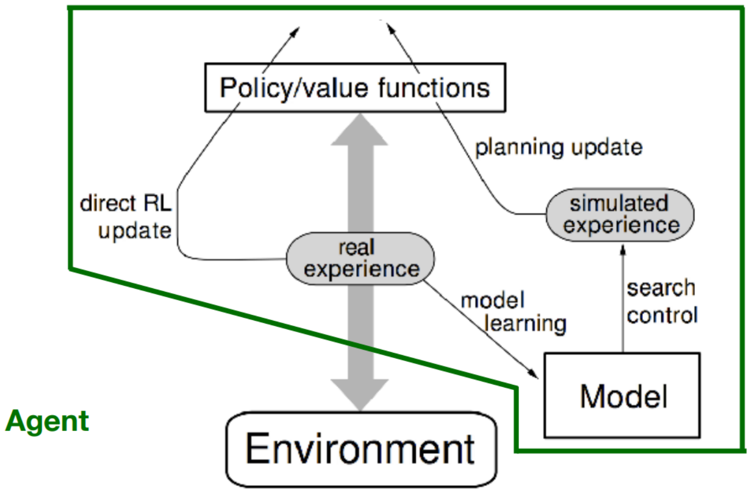
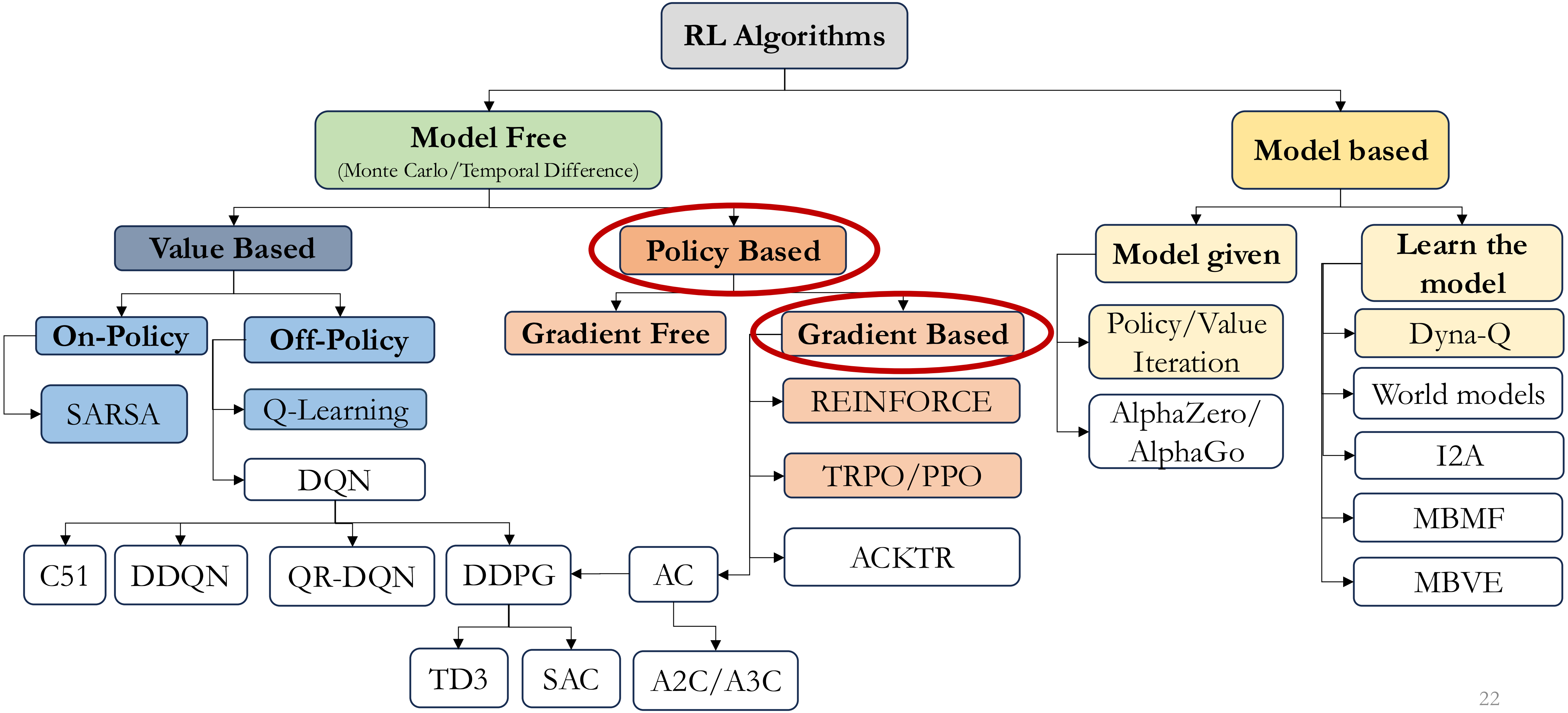


Image Credit: Sutton and Barto

Tabular Dyna-Q

- Initialize $\hat{q}(s, a)$ and $\mathcal{M}(s, a)$ $\forall s \in \mathcal{S}, a \in \mathcal{A}$
- Loop
  - Sample current state S_t
  - Sample action A_t at S_t based on ϵ-greedy on $\hat{q}$
  - Take action A_t, observe R_{t+1} and S_{t+1}
  - $\hat{q}(S_{t+1}, A_t) \leftarrow \hat{q}(S_t, A_t) + \alpha([R_{t+1} + \gamma \max_{A'} \hat{q}(S_{t+1}, A')] - \hat{q}(S_t, A_t))$
  - $\mathcal{M}(S_t, A_t) \leftarrow R_{t+1}, S_{t+1}$
  - Loop n times
    - Sample random state s
    - Sample random previous action a at s
    - $r, s' \leftarrow \mathcal{M}(s, a)$
    - $\hat{q}(s, a) \leftarrow \hat{q}(s, a) + \alpha([r + \gamma \max_{a'} \hat{q}(s', a')] - \hat{q}(s, a))$

Categorizing RL Algorithms



Policy-Based RL – Policy Gradient Methods

- Previously, we approximated the value functions using parameters $\boldsymbol{\theta}$
  - Obtained policy from value function $\hat{v}(s, \boldsymbol{\theta})$ or $\hat{q}(s, a, \boldsymbol{\theta})$ using ϵ-greedy
- Now, directly parameterize and learn the policy $\pi_\theta(s, a) = \mathbb{P}[a|s, \boldsymbol{\theta}]$
  - Model-Free RL, better convergence properties, can learn stochastic policies
  - Effective in high-dimensional or continuous action spaces
  - Typically converge to a local rather than global optimum
  - Evaluating a policy is typically inefficient and high variance
- Given a policy $\pi_\theta(s, a)$ with parameters $\boldsymbol{\theta}$, find best $\boldsymbol{\theta}$ which maximizes $J(\boldsymbol{\theta})$
  - **Policy Objective Function** $J(\boldsymbol{\theta})$ - Measures quality of policy π_θ
    - Episodic environments: $J(\boldsymbol{\theta}) = v_{\pi_\theta}(s_1, \boldsymbol{\theta})$(also called start value)
    - Continuing environments: $J(\boldsymbol{\theta}) = \sum_s d_{\pi_\theta}(s) v_{\pi_\theta}(s, \boldsymbol{\theta})$ (also called average value), $d_{\pi_\theta}(s)$ is the stationary distribution of Markov chain for π_θ
- Can use gradient free optimization, but greater efficiency possible using gradient
- Policy Gradient Methods:
  - Search for local maximum by **ascending** the policy gradient with $\boldsymbol{\theta}$: $\Delta\boldsymbol{\theta} = \boldsymbol{\alpha} \boldsymbol{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

(6.B) Monte Carlo Policy Gradient – REINFORCE

- Policy Gradient Theorem
  - For any differentiable policy

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_\theta(S_t, A_t)\, q_{\pi_\theta}(S_t, A_t)\right]$$

  - $\nabla_{\boldsymbol{\theta}} \log \pi_\theta(s, a)$ is called the **score function**
  - Key observations
    - It allows gradients of policy instead of value.
    - The action value q_{π_θ} can be approximated.
  - Many choices of differentiable policy π_θ – Softmax, Gaussian, Neural Networks
- Monte Carlo Policy Gradient
  - Update parameters by stochastic gradient ascent, use policy gradient theorem
  - Use return G_t as an unbiased estimate of $q_{\pi_\theta}(S_t, A_t)$
  - $\Delta\boldsymbol{\theta} = \alpha \nabla_{\boldsymbol{\theta}} \log \pi_\theta(S_t, A_t)\, G_t$
- MC policy gradient has high variance
  - Use actor-critic methods to reduce variance

REINFORCE

- Initialize policy parameters $\boldsymbol{\theta}$ arbitrarily
- Loop
  - Sample episode following policy π_θ $(S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T)$
  - For $t = 1$ to $T - 1$
    - $G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots \gamma^{T-1} R_T$
    - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} \log \pi_\theta(S_t, A_t)\, G_t$
- Return $\boldsymbol{\theta}$

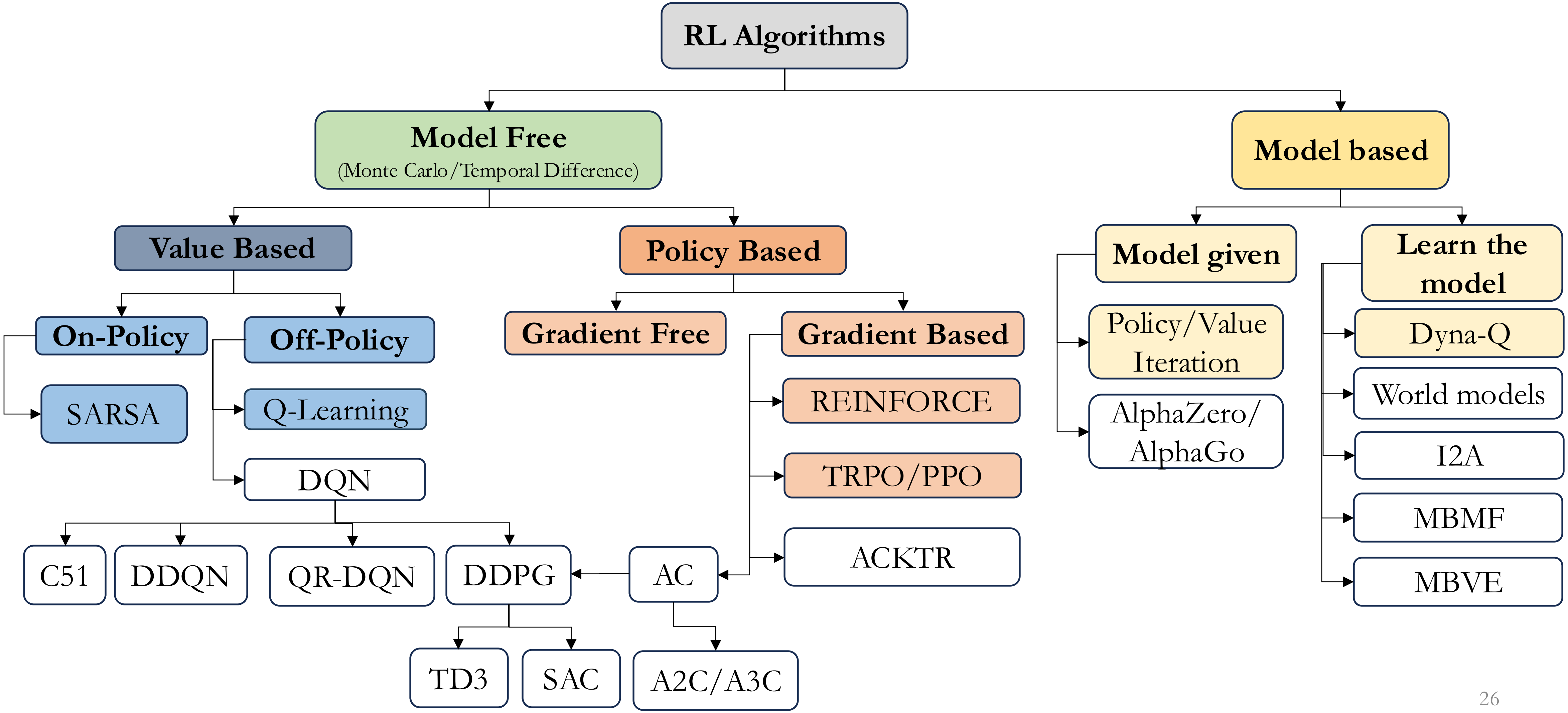
(7.B) Advanced Policy Gradient Algorithms – Trust Region Methods (TRPO/PPO)

- General policy gradient algorithms try to solve the optimization problem

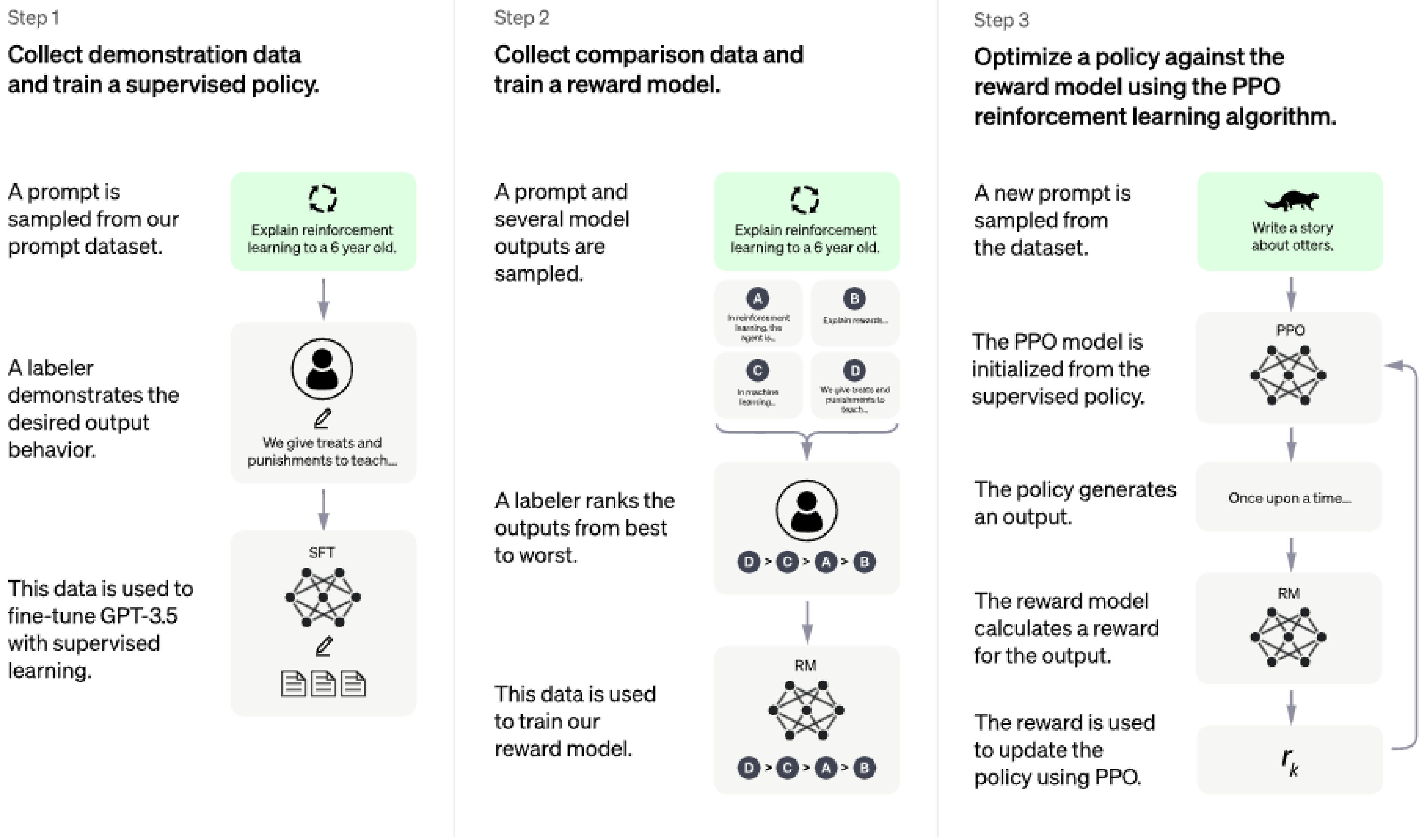
$$\max_{\boldsymbol{\theta}} J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t R_t]$$

- Use stochastic gradient ascent on policy parameters $\boldsymbol{\theta}$ using policy gradient g
  - $g = \nabla_{\boldsymbol{\theta}} J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t \nabla_{\boldsymbol{\theta}} \log \pi_\theta(A_t|S_t) \mathbf{A}_{\pi_\theta}(S_t, A_t)]$
  - Advantage function $\mathbf{A}_{\pi_\theta}(\boldsymbol{s}, \boldsymbol{a}) = \boldsymbol{q}_{\pi_\theta}(\boldsymbol{s}, \boldsymbol{a}) - \boldsymbol{v}_{\pi_\theta}(\boldsymbol{s})$, relative **advantage** of an action i.e. how much better to take action $\boldsymbol{a}$ in state $\boldsymbol{s}$ over randomly selecting any other action and following π_θ after
- However, its sample efficiency is poor as it searches in **parameter space** instead of policy space. Also, the method is dependent on step size.
- Trust Region Methods – Proximal Policy Optimization(PPO)
  - Define $\mathcal{L}_\pi(\pi') \approx J(\pi') - J(\pi)$ ($\pi' \rightarrow$ new policy, $\pi \rightarrow$ old policy), improvement over old policy
  - Update $\boldsymbol{\theta}$ incrementally, approximately penalize policies from changing too much between steps
  - **Adaptive KL Penalty**: $\boldsymbol{\theta}_{k+1} = \operatorname{argmax}_{\boldsymbol{\theta}} \mathcal{L}_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}) - \beta_k KL(\boldsymbol{\theta}||\boldsymbol{\theta}_k)$, β_k is the penalty coefficient
  - **Clipped Objective**: $\boldsymbol{\theta}_{k+1} = \operatorname{argmax}_{\boldsymbol{\theta}} \mathcal{L}^{CLIP}_{\boldsymbol{\theta}_k}(\boldsymbol{\theta})$ where
    $\mathcal{L}^{CLIP}_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim\pi_k}[\sum_{t=0}^{T}[\min(r_t(\boldsymbol{\theta})\hat{\mathbf{A}}_{\pi_k}(S_t, A_t), \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{\mathbf{A}}_{\pi_k}(S_t, A_t))]]$,
    $r_t(\boldsymbol{\theta}) = \pi_\theta(A_t|S_t) / \pi_{\theta_k}(A_t|S_t)$, ϵ is a hyperparameter

Categorizing RL Algorithms



RL Application: Reinforcement Learning using Human Feedback - Finetuning ChatGPT



Example copied verbatim from https://openai.com/blog/chatgpt.

Summary of RL Algorithms

- Agent attempts to find **optimal policies** with highest returns via. environment interaction
  - **Planning/Prediction** evaluates a given policy and **Learning/Control** finds the optimal policy
  - Policy Iteration for control involves value function estimation and policy improvement steps
- Model-Free learning does not require model of the environment (MDP)
  - **Monte Carlo (MC)** estimates the future returns by sampling returns via. environment interaction
  - **Temporal Difference (TD)** estimates the future returns in a more online manner
  - **SARSA (On-policy)** and **Q-Learning (off-policy)** uses MC/TD for model-free control
- Model-Based learning like **Dyna-Q** estimates the model of the environment (MDP)
- The **<u>state-value</u>**, **<u>action-value functions</u>** and **<u>policies</u>** can be approximated for large MDPs using neural networks or other parametric function approximators
- Policy gradient methods directly find optimal policies using gradient descent
- In practice, RL algorithms can be used in various applications like stock trading, self-driving cars and even systems like ChatGPT

References

- Based on the excellent RL book by Sutton and Barto
  - http://incompleteideas.net/book/the-book-2nd.html
- Some content borrowed from David Silver's Lecture Notes
  - https://www.davidsilver.uk/teaching/
- Additional help from Stanford CS234 course by Emma Brunskill
  - https://web.stanford.edu/class/cs234/modules.html
- OpenAI Blogs
  - https://openai.com/blog/chatgpt
  - https://spinningup.openai.com/en/latest/index.html