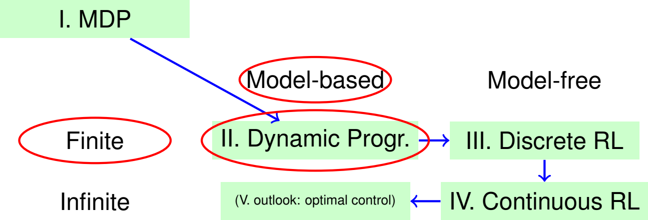
# AIC/RL – Discrete Reinforcement Learning (Part III) Temporal Differencing

Freek Stulp

Université Paris-Saclay

# Where are we?



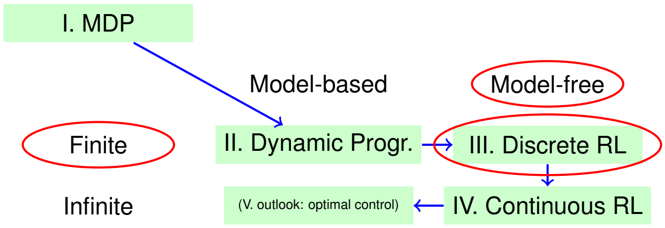

## Dynamic Programming (Part II) in three formulae

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma V_k(s') \right] \quad \text{Policy evaluation} \tag{1}$$

$$\pi(s) = \operatorname*{argmax}_a \sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma V(s') \right] \quad \text{Policy improvement} \tag{2}$$

$$V_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma V_k(s') \right] \quad \text{Value iteration} \tag{3}$$

# Where are we?




## Important topics in Part III

- Update values from observations...
  - **Monte-Carlo**: from returns (i.e. at end of each episode)
  - **Temporal Differencing**: from immediate rewards (i.e. after each action)
- Learning **state/action values** $Q$ instead of state values $V$
- Should I learn better values, or exploit the ones I have?
  - **Exploration/Exploitation trade-off**

# Model-based vs. Model-free

- Environment as an MDP: $\{S, A, \mathcal{P}, \mathcal{R}\}$
  - $S$ Possible states
  - $A$ Possible actions
  - $\mathcal{P}$ Transition function
  - $\mathcal{R}$ Reward function

## Model-based

- Agent knows $\mathcal{P}$ and $\mathcal{R}$
  - Can use $\mathcal{P}$/$\mathcal{R}$ to compute values
  - Dynamic Programming
- Compute values "in your head"

## Model-free

- Agent does not know $\mathcal{P}$ and/or $\mathcal{R}$
  - Cannot do Dyn. Prog.
    $V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right]$
- Estimate values from direct interaction with the environment
  - Only rely on actual observations
  - What you do influences what you see

# Model-based vs. Model-free

## Important topics in Part III

- Update values from observations...
  - **Monte-Carlo**: from returns (i.e. at end of each episode)
  - **Temporal Differencing**: from immediate rewards (i.e. after each action)
- Learning **state/action values** $Q$ instead of state values $V$
- Should I learn better values, or exploit the ones I have?
  - **Exploration/Exploitation trade-off**

## Model-based

- Agent knows $\mathcal{P}$ and $\mathcal{R}$
  - Can use $\mathcal{P}/\mathcal{R}$ to compute values
  - Dynamic Programming
- Compute values "in your head"

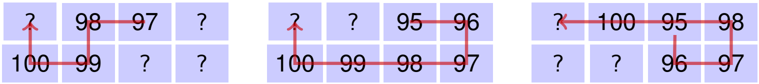
## Model-free

- Agent does not know $\mathcal{P}$ and/or $\mathcal{R}$
  - Cannot do Dyn. Prog.

    $V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')\right]$
- Estimate values from direct interaction with the environment
  - Only rely on actual observations
  - What you do influences what you see

# Monte Carlo Policy Evaluation (SUBA5.1)

- Example: random policy, deterministic gridworld, start in $s_2$




$R(s_2)^1 = 97$ $R(s_2)^2 = 95$ $R(s_2)^3 = 95$

- Given these episodes, what is $V^{\pi}(s_2) = \mathrm{E}_{\pi}\left\{R_t | s_t = s_2\right\}$ ?
  - Even if we don't have $\mathcal{P}$ or $\mathcal{R}$, we <u>can</u> estimate it given these observations!

# Putting it all together

RL Superman: uses Monte-Carlo to estimate state-action values using $\epsilon$-greedy exploration with decaying $\epsilon$

- Why super?
    - Learns $Q$-values from observed returns (doesn't require a model)
    - Estimates become better over time with more experience
    - Can choose the best action as $\text{argmax}_a Q^\pi(s, a)$
    - Starts out with exploration ($\epsilon = 1$), but slowly becomes greedy ($\epsilon \approx 0$)
- But...
    - requires a lot of experience to get good estimates and policy
    - works for small finite MDPs only
    - applicable to episodic problems only
    - wastes time evaluating bad policies
- Solution: Temporal Difference Learning

# Temporal Differencing (SuBa6)

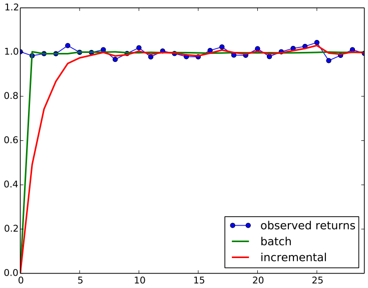
*"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning."* (SuBa6)

- Monte Carlo
  - update values after an episode is done (based on returns $R = \sum_t r_t$
- Temporal-difference learning
  - update values after each step (based on immediate reward $r_t$)

- Explained in two steps
  1. Show incremental version of Monte Carlo (uses $R$)
  2. Show TD learning (uses $r_t$)

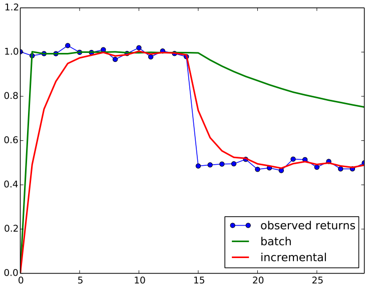
# Monte Carlo: Batch vs. Incremental

- Batch: $V^\pi(s) = average(Returns(s)) = \frac{1}{N}\sum_{e=1}^{N} R(s)^e$
  - $N$ is the number of episodes
- Incremental updates: $V^\pi(s) = V^\pi(s) + \alpha\,[R - V^\pi(s)]$
  - $0 < \alpha < 1$ is 'learning rate' ($\alpha = 1$ would be batch, $\alpha = 0$ would mean no learning)
  - $\alpha$ also known as 'step-size'



# Monte Carlo: Batch vs. Incremental

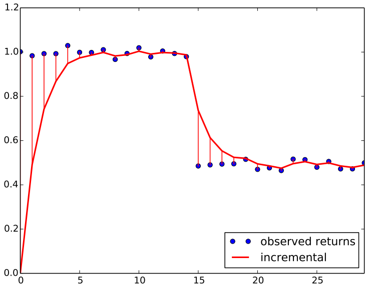
- Batch: $V^{\pi}(s) = average(Returns(s)) = \frac{1}{N}\sum_{e=1}^{N} R(s)^e$
  - $N$ is the number of episodes
- Incremental updates: $V^{\pi}(s) = V^{\pi}(s) + \alpha\,[R - V^{\pi}(s)]$
  - $0 < \alpha < 1$ is 'learning rate' ($\alpha = 1$ would be batch, $\alpha = 0$ would mean no learning)
  - $\alpha$ also known as 'step-size'




- Incremental better if environment is non-stationary (changes over time)

# Monte Carlo: Batch vs. Incremental

- Batch: $V^\pi(s) = average(Returns(s)) = \frac{1}{N}\sum_{e=1}^{N} R(s)^e$
  - $N$ is the number of episodes
- Incremental updates: $V^\pi(s) = V^\pi(s) + \alpha\,[R - V^\pi(s)]$
  - $0 < \alpha < 1$ is 'learning rate' ($\alpha = 1$ would be batch, $\alpha = 0$ would mean no learning)
  - $\alpha$ also known as 'step-size'



# From incremental MC to TD (SuBa6.1)

Definition

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\{R_t | s_t = s\} \\
&= E_{\pi}\{\textstyle\sum_{k=0}^{T} \gamma^k r_{t+k+1} | s_t = s\} \\
&= E_{\pi}\{r_{t+1} + \gamma \textstyle\sum_{k=1}^{T} \gamma^{k-1} r_{t+k+1} | s_t = s\} \\
&= E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s\}
\end{aligned}
$$

Bellman equation

Incremental estimation from experience

$$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha[R_t - V(s_t)] \quad \text{(MC)}$$

$$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad \text{(TD)}$$

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

Figure : Policy evaluation with $TD(0)$.

# From incremental MC to TD (SuBa6.1)

Definition

$$
\begin{aligned}
V^\pi(s) &= E_\pi\{R_t | s_t = s\} \\
&= E_\pi\{\textstyle\sum_{k=0}^{T} \gamma^k r_{t+k+1} | s_t = s\} \\
&= E_\pi\{r_{t+1} + \gamma \textstyle\sum_{k=1}^{T} \gamma^{k-1} r_{t+k+1} | s_t = s\} \\
&= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s\}
\end{aligned}
$$

Bellman equation

Incremental estimation from experience

$$V^\pi(s_t) = V^\pi(s_t) + \alpha[R_t - V(s_t)] \quad \text{(MC)}$$

$$V^\pi(s_t) = V^\pi(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad \text{(TD)}$$

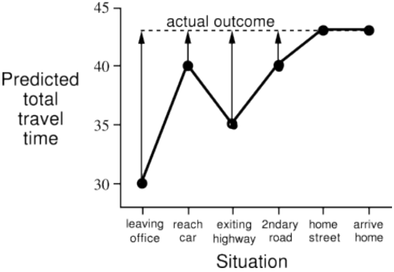



Figure : Driving home (MC)

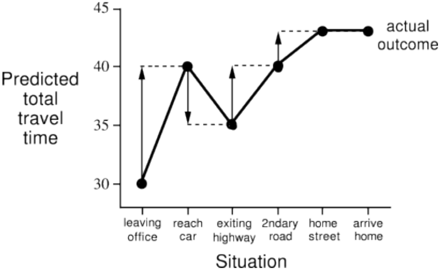



Figure : Driving home (TD)

# TD learning for $Q$-Values (SARSA) (SuBa6.4)

$$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{1}$$

$$Q^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2}$$

- Incremental update requires $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$
  - Known as “SARSA”
  - If $s_{t+1}$ is terminal, then $Q(s_{t+1}, a_{t+1})$ is zero

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma Q(s', a') - Q(s, a)\big]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until $s$ is terminal

Figure : SARSA: an on-policy TD control algorithm

# Summary of (model-free) discrete RL

- Model-free learning for state values

| | Definition | Estimation from observed experience: Batch | Incremental | |
|---|---|---|---|---|
| $V^\pi(s)$ | $= E_\pi\{R_t \| s_t = s\}$ | $\frac{1}{N}\sum_{e=1}^{N} R(s)^e$ | $V^\pi(s_t) = V^\pi(s_t) + \alpha[R_t - V(s_t)]$ | MC |
| | $= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \| s_t = s\}$ Bellman equation | | $V^\pi(s_t) = V^\pi(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$ | TD |

- Model-free learning for state/action values

| | Definition | Estimation from observed experience: Batch | Incremental | |
|---|---|---|---|---|
| $Q^\pi(s,a)$ | $= E_\pi\{R_t \| s_t = s, a_t = a\}$ | $\frac{1}{N}\sum_{e=1}^{N} R(s,a)^e$ | $Q^\pi(s_t,a_t) = Q^\pi(s_t,a_t) + \alpha[R_t - Q(s_t,a_t)]$ | MC |
| | $= E_\pi\{r_{t+1} + \gamma Q^\pi(s_{t+1},a_{t+1}) \| s_t = s, a_t = a\}$ Bellman equation | | $Q^\pi(s_t,a_t) = Q^\pi(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$ | SARSA |

# DYNA (SuBa9.2)

- How about using experience to not only update values $Q(s, a)$, but also a model $P^a_{ss'}$ ?

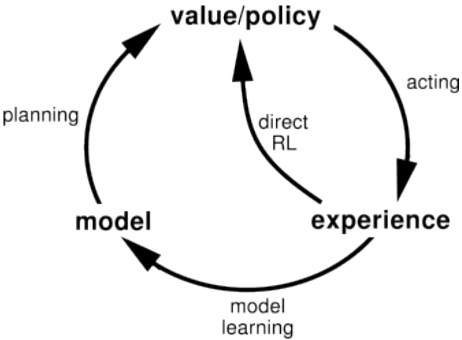



Figure : Relationships among learning, planning, and acting

# DYNA (SuBa9.2)

- How about using experience to not only update values $Q(s, a)$, but also a model $P^a_{ss'}$ ?

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
 (a) $s \leftarrow$ current (nonterminal) state
 (b) $a \leftarrow \varepsilon$-greedy$(s, Q)$
 (c) Execute action $a$; observe resultant state, $s'$, and reward, $r$
 (d) $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
 (e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment)
 (f) Repeat $N$ times:
  $s \leftarrow$ random previously observed state
  $a \leftarrow$ random action previously taken in $s$
  $s', r \leftarrow Model(s,a)$
  $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

Figure : Dyna-Q Algorithm

# DYNA (SuBa9.2)

- How about using experience to not only update values $Q(s, a)$, but also a model $P^a_{ss'}$ ?

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

(a) $s \leftarrow$ current (nonterminal) state

(b) $a \leftarrow \varepsilon$-greedy$(s, Q)$

**Update $Q$ with real experience**

(c) Execute action $a$; observe resultant state, $s'$, and reward, $r$

(d) $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big]$

(e) $Model(s, a) \leftarrow s', r$ (assuming deterministic environment)

(f) Repeat $N$ times:

$s \leftarrow$ random previously observed state

$a \leftarrow$ random action previously taken in $s$

$s', r \leftarrow Model(s, a)$

$Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\big]$

Figure : Dyna-Q Algorithm

# DYNA (SuBa9.2)

- How about using experience to not only update values $Q(s,a)$, but also a model $P^a_{ss'}$ ?

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
(a) $s \leftarrow$ current (nonterminal) state
(b) $a \leftarrow \varepsilon$-greedy$(s, Q)$
(c) Execute action $a$; observe resultant state, $s'$, and reward, $r$
**Update model with real experience**
(d) $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
(e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment)
(f) Repeat $N$ times:
$s \leftarrow$ random previously observed state
$a \leftarrow$ random action previously taken in $s$
$s', r \leftarrow Model(s,a)$
$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

Figure : Dyna-Q Algorithm

# DYNA (SuBa9.2)

- How about using experience to not only update values $Q(s,a)$, but also a model $P^a_{ss'}$ ?

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

(a) $s \leftarrow$ current (nonterminal) state

(b) $a \leftarrow \varepsilon$-greedy$(s,Q)$

(c) Execute action $a$; observe resultant state, $s'$, and reward, $r$

(d) $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$

(e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment)

(f) Repeat $N$ times:

**Generate simulated experience with model**

$s \leftarrow$ random previously observed state

$a \leftarrow$ random action previously taken in $s$

$s', r \leftarrow Model(s,a)$

$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$

Figure : Dyna-Q Algorithm

# DYNA (SuBa9.2)

- How about using experience to not only update values $Q(s, a)$, but also a model $P^a_{ss'}$ ?

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
(a) $s \leftarrow$ current (nonterminal) state
(b) $a \leftarrow \varepsilon$-greedy$(s, Q)$
(c) Execute action $a$; observe resultant state, $s'$, and reward, $r$
(d) $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
(e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment)
(f) Repeat $N$ times:
$s \leftarrow$ random previously observed state
$a \leftarrow$ random action previously taken in $s$
$s', r \leftarrow Model(s,a)$
$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$

**Update $Q$ with with simulated experience**

Figure : Dyna-Q Algorithm

# DYNA (SuBa9.2)

- How about using experience to not only update values $Q(s, a)$, but also a model $P^a_{ss'}$ ?

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
(a) $s \leftarrow$ current (nonterminal) state
(b) $a \leftarrow \varepsilon$-greedy$(s, Q)$
**TD learning**
(c) Execute action $a$; observe resultant state, $s'$, and reward, $r$
(d) $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
(e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment)
(f) Repeat $N$ times:
$s \leftarrow$ random previously observed state
$a \leftarrow$ random action previously taken in $s$
$s', r \leftarrow Model(s,a)$
$Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$

**Dynamic Programming**

Figure : Dyna-Q Algorithm

# DYNA (SuBa9.2)

- How about using experience to not only update values $Q(s, a)$, but also a model $P^a_{ss'}$ ?

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
 (a) $s \leftarrow$ current (nonterminal) state
 (b) $a \leftarrow \varepsilon$-greedy$(s, Q)$
 (c) Execute action $a$; observe resultant state, $s'$, and reward, $r$

**TD learning (Like real life experience?)**

 (d) $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
 (e) $Model(s,a) \leftarrow s', r$ (assuming deterministic environment)
 (f) Repeat $N$ times:
  $s \leftarrow$ random previously observed state
  $a \leftarrow$ random action previously taken in $s$
  $s', r \leftarrow Model(s,a)$
  $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$

**Dynamic Programming (Like dreams?)**

Figure : Dyna-Q Algorithm