Lecture 11: Policy learning

Jochen Braun

Otto-von-Guericke-Universität Magdeburg,
Cognitive Biology Group

Theoretical Neuroscience II, SS 2020

Credits:
Dayan & Abbott (2001) "Theoretical Neuroscience", Chapter 9
Sutton & Barto (1998) "Reinforcement Learning", Chapters 3-7

11. Policy learning

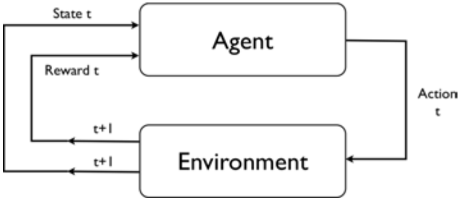
*Active agents must learn action preferences (***'policies'***). Learning may be based on immediate rewards ('static action choice') or delayed rewards ('sequential action choice'). Main principles illustrated by* **bees foraging for nectar***. Visit previously rewarding flowers, or try new ones (***exploration-exploitation***)? Act smartly on basis of reward expectations (***indirect actor*** *with "model"), or greedily on basis of rewards (***direct actor*** *without "model"). Return to* **TDL for continuous tasks***, with 'gridworld' and 'agent' seeking exit. Aim to learn both value function* $V_\pi(s)$ *and policy* $\pi(a,s)$ *for this situation. Theoretical approach (***policy iteration)*** *reveals difficulties: improving* $V_\pi(s)$ *invalidates* $\pi(a,s)$ *(and vice versa). Biological approach (***TDL***) is simple and effective, transforming 'gridworld' into veritable paradise.*

Outline

1 Policy learning

In nature, rewards are more often associated with actions taken than with stimuli observed. Animals are very adept at learning policies (action preferences) to increase rewards.

First we consider **static action choice**, where correct actions are rewarded **immediately**.



At the end of the lecture, we will consider **sequential action choice**, where rewards may be **delayed** until several actions have been taken.

Bee foraging

An example of static action choice are bees foraging among flowers in search for nectar. Assume that single bees forage under controlled conditions among blue and yellow “flowers” (small dishes of sugar water). After some dozens of visits to different blue and yellow “flowers”, bees learn the reward characteristics and preferentially visit the more rewarding color. If reward patterns change, bees quickly adapt and change their preferences.

Explorative and exploitative policies

Assume that bees visit blue or yellow flowers with probability P_b and P_y, respectively. These values constitute a *policy* $\pi(a)$

$$\pi(a) = \{P_b, P_a\} \qquad\qquad P_b + P_y = 1$$

and succinctly summarize the bee's preference (perhaps based on prior experience).

Exploratory policy: no preference between previously rewarded and unrewarded actions, reveals an adventurous disposition.

$$P_b \sim P_y \sim 0.5$$

Exploitative policy: strong preference for previously rewarded actions, reveals reliance on past experience.

$$P_b \sim 1 \qquad P_y \sim 0$$

Action values

We summarize prior experience in terms of *action values* m_b and m_y and derive action probabilities with a **softmax** formula:

$$P_b = \frac{1}{1 + \exp\left[\beta\left(m_y - m_b\right)\right]} \qquad P_y = \frac{1}{1 + \exp\left[\beta\left(m_b - m_y\right)\right]}$$

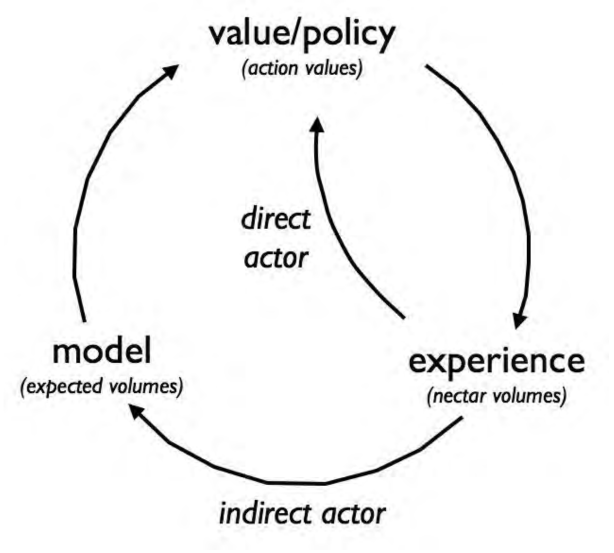
$$P_b + P_y = 1$$

The constant β determines the variability of the bee's choice. A small β makes random *exploration* more frequent. A large β makes deterministic *exploitation* more likely.

Action values may either be just placeholders or may represent reward expectations.

Direct and indirect actors

Is it better to act greedily, or is it better to act smartly?



2 Indirect actor

In the ‘indirect actor’ approach, the policy is based on learning a ‘model’, which here consists of expected rewards:

$$m_b = \langle r_b \rangle \qquad\qquad m_y = \langle r_y \rangle$$

Action values correspond to reward expectations and may be learned with the Rescorla-Wagner rule:

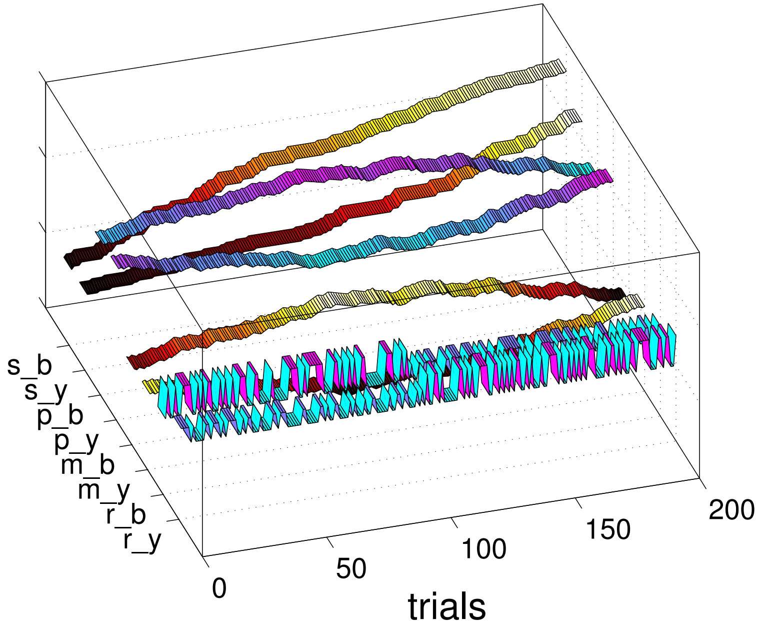
Choose blue:

$$m_b \to m_b + \epsilon\, \delta_b \qquad\qquad \delta_b = r_b - m_b$$
$$m_y \to m_y$$

Choose yellow:

$$m_y \to m_y + \epsilon\, \delta_y \qquad\qquad \delta_y = r_y - m_y$$
$$m_b \to m_b$$

Exploitation/exploration $\beta = 2$



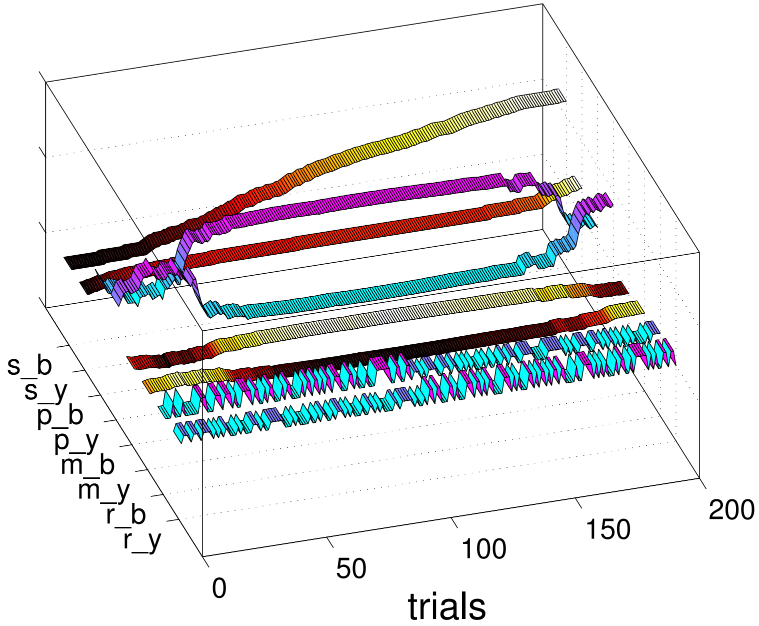
Exploitation/exploration $\beta = 2$

The previous page shows representative model performance for a more **'exploratory'** value of $\beta = 2$. Reward pattern changes after trial 100!

- $r_{y,b}$ represent (partly stochastic) rewards obtained over 200 trials.
- $m_{y,b}$ represent action values accumulated with RW rule.
- $p_{y,b}$ represent action probabilities derived from $m_{y,b}$ with **softmax** formula.
- $s_{y,b} = \sum r_{y,b}$ represent cumulative rewards obtained up to each trial.

The model benefits moderately from both reward patterns.

Exploitation/exploration $\beta = 20$



Exploitation/exploration $\beta = 20$

The previous page shows representative model performance for a more **'exploitative'** value of $\beta = 20$. Reward pattern changes after trial 100!

- $r_{y,b}$ represent (partly stochastic) rewards obtained over 200 trials.
- $m_{y,b}$ represent action values accumulated with RW rule.
- $p_{y,b}$ represent action probabilities derived from $m_{y,b}$ with **softmax** formula.
- $s_{y,b} = \sum r_{y,b}$ represent cumulative rewards obtained up to each trial.

The model initially benefits greatly, but then gets stuck in an inappropriate policy.

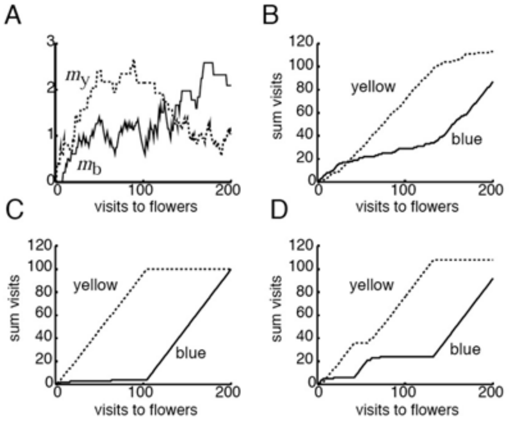
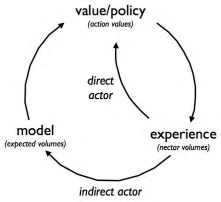


Figure 9.4: The indirect actor. Rewards were $\langle r_b \rangle = 1$, $\langle r_y \rangle = 2$ for the first 100 flower visits, and $\langle r_b \rangle = 2$, $\langle r_y \rangle = 1$ for the second 100 flower visits. Nectar was delivered stochastically on half the flowers of each type. A) Values of m_b (solid) and m_y (dashed) as a function of visits for $\beta = 1$. Because a fixed value of $\epsilon = 0.1$ was used, the weights do not converge perfectly to the corresponding average reward, but they fluctuates around these values. B-D) Cumulative visits to blue (solid) and yellow (dashed) flowers. B) When $\beta = 1$, learning is slow, but ultimately the change to the optimal flower color is made reliably. C;D) When $\beta = 50$, sometimes the bee performs well (C), and other times it performs poorly (D).

Points to note

- Some actors are **smart** and first learn reward expectations (nectar volumes).
- Reward expectations serve as *action values*, which later inform action probabilities.
- Reward expectations are changed only for actions actually taken (Rescorla-Wager).
- Softmax parameter β controls exploitation-exploration balance.
- These *indirect actors* base actions on world model (reward expectations).



3 Direct actor

Alternatively, one may maximize expected reward directly, without bothering to learn the reward expectations of individual actions (i.e., without learning a “model”). The dependence of reward expecation

$$\langle r \rangle = P_b \, \langle r_b \rangle + P_y \, \langle r_y \rangle$$

on action values is as follows:

$$\frac{\partial \langle r \rangle}{\partial m_b} = \beta \, P_b \, P_y \, \left(\langle r_b \rangle - \langle r_y \rangle \right), \qquad \frac{\partial \langle r \rangle}{\partial m_y} = \beta \, P_b \, P_y \, \left(\langle r_y \rangle - \langle r_b \rangle \right)$$

ctd

Note that

$$P_{b,y} = \frac{1}{1+\exp\left[\beta\left(m_{y,b} - m_{b,y}\right)\right]} = \frac{\exp\left[\beta\left(m_{b,y} - m_{y,b}\right)\right]}{1+\exp\left[\beta\left(m_{b,y} - m_{y,b}\right)\right]}$$

implies

$$\frac{\partial P_b}{\partial m_b} = \frac{\partial P_y}{\partial m_y} = \beta\, P_b\, P_y \qquad \frac{\partial P_y}{\partial m_b} = \frac{\partial P_b}{\partial m_y} = -\beta\, P_b\, P_y$$

so that

$$\frac{\partial \langle r \rangle}{\partial m_b} = \langle r_b \rangle \frac{\partial P_b}{\partial m_b} + \langle r_y \rangle \frac{\partial P_y}{\partial m_b} = \beta\, P_b\, P_y\, \left(\langle r_b \rangle - \langle r_y \rangle\right)$$

$$\frac{\partial \langle r \rangle}{\partial m_y} = \langle r_b \rangle \frac{\partial P_b}{\partial m_y} + \langle r_y \rangle \frac{\partial P_y}{\partial m_y} = \beta\, P_b\, P_y\, \left(\langle r_y \rangle - \langle r_b \rangle\right)$$

Stochastic gradient ascent

$\langle r \rangle$ may be maximized by choosing increments Δm_b and Δm_y which are (on average) proportional to $\delta\langle r\rangle/\delta m_b$ and $\delta\langle r\rangle/\delta m_y$.

A suitable gradient ascent rule is (see D&A):

Choose blue:

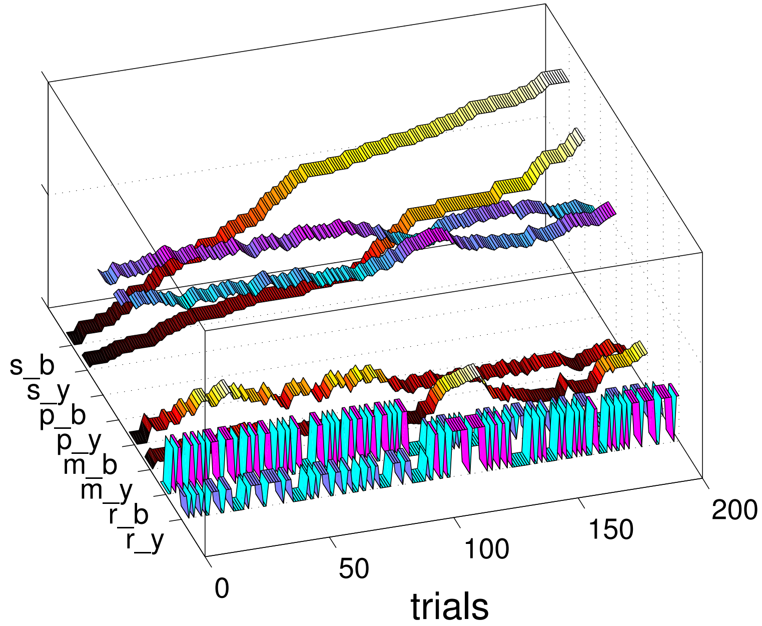
$$\begin{aligned} m_b &\rightarrow m_b + \epsilon\,(1 - P_b)\,(r_b - \bar{r}) \\ m_y &\rightarrow m_y - \epsilon\, P_y\,(r_b - \bar{r}) \end{aligned}$$

Choose yellow:

$$\begin{aligned} m_y &\rightarrow m_y + \epsilon\,(1 - P_y)\,(r_y - \bar{r}) \\ m_b &\rightarrow m_b - \epsilon\, P_b\,(r_y - \bar{r}) \end{aligned}$$

where $\bar{r}$ is the average reward.

Exploitation/exploration $\beta = 2$



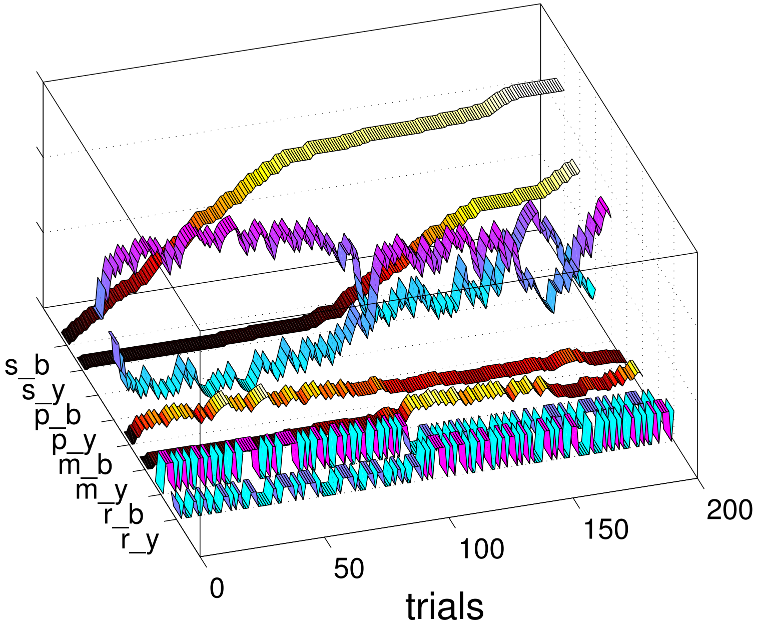
Exploitation/exploration $\beta = 2$

The previous page shows representative model performance for a more **'exploratory'** value of $\beta = 20$. Reward pattern changes after trial 100!

- $r_{y,b}$ represent (partly stochastic) rewards obtained over 200 trials.
- $m_{y,b}$ represent action values accumulated with gradient ascent.
- $p_{y,b}$ represent action probabilities derived from $m_{y,b}$ with **softmax** formula.
- $s_{y,b} = \sum r_{y,b}$ represent cumulative rewards obtained up to each trial.

The model benefits moderately from both reward patterns, but sometimes gets stuck in inappropriate policy.

Exploitation/exploration $\beta = 20$



Exploitation/exploration $\beta = 20$

The previous page shows representative model performance for a more **‘exploitative’** value of $\beta = 20$. Reward pattern changes after trial 100!

- $r_{y,b}$ represent (partly stochastic) rewards obtained over 200 trials.
- $m_{y,b}$ represent action values accumulated with gradient ascent.
- $p_{y,b}$ represent action probabilities derived from $m_{y,b}$ with **softmax** formula.
- $s_{y,b} = \sum r_{y,b}$ represent cumulative rewards obtained up to each trial.

The model benefits greatly from both reward patterns and rarely gets stuck in an inappropriate policy.

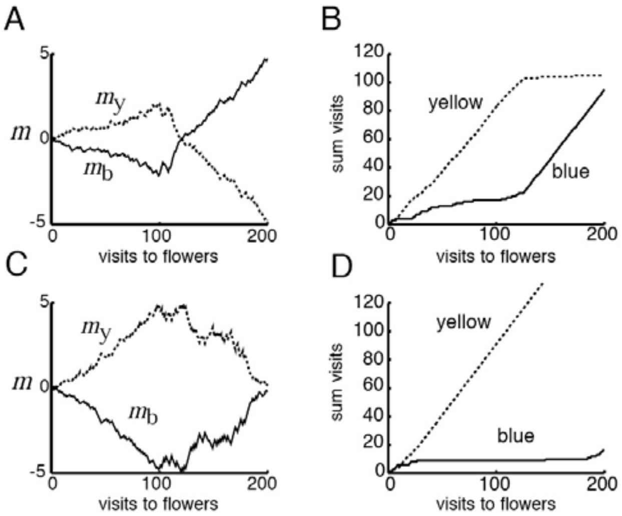
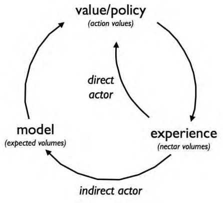


Figure 9.6: The direct actor. The statistics of the delivery of reward are the same as in figure 9.4, and $\epsilon = 0.1$, $\bar{r} = 1.5$, and $\beta = 1$. The evolution of the weights and cumulative choices of flower type (with yellow dashed and blue solid) are shown for two sample sessions, one with good performance (A & B) and one with poor performance (C & D).

Points to note

- Some actors are *greedy* and base action values on reward received (without learning reward expectations).
- Action values are incremented for all actions, taken or not, and have arbitrary units.
- Softmax parameter β controls exploitation/exploration balance.
- These *direct actors* operate without a world model (reward expectations).



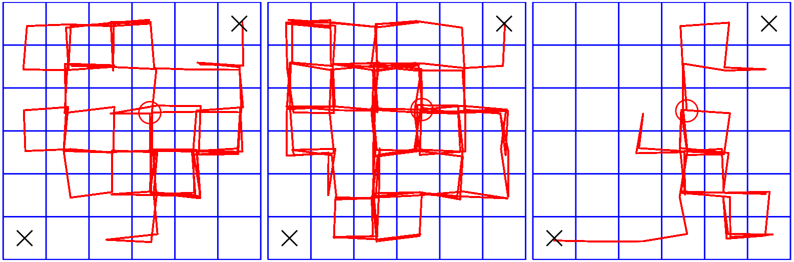
Comparing *direct* and *indirect* actors

- *Indirect* schemes are more complex, but sometimes learn more efficiently.
- Success depends on appropriateness of adopted 'world model'.

- *Direct* schemes are simpler, but sometimes learn less efficiently.
- Success is independent of 'model bias'.

4 Gridworld with policy iteration (mere theory)

We return to our "gridworld" of $N \times N$ states, in which an agent moves randomly until it accidentally finds one of two exits:



Random policy

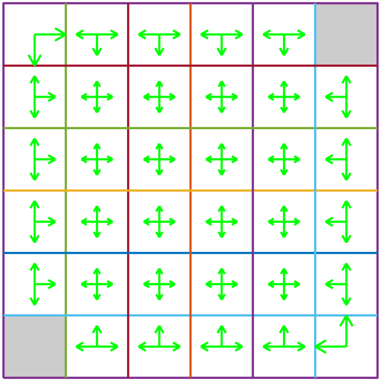
A policy π is a mapping from states s to actions a:

$$\pi(a, s) \quad \text{probability of transition} \quad s \xrightarrow{a} s'$$

So far, our policy has been random:

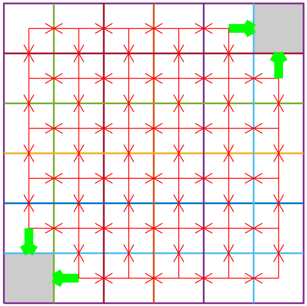
$$\pi(a|s) = {}^{1}/_{N_a(s)}$$

where $N_a(s)$ is the number of available actions from state s.



Reinforcement structure

The gridworld architect imposes a certain reinforcement structure (for all states and actions). Here, we choose $+10$ for exit steps and -1 for all other steps.



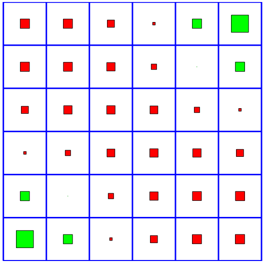
An animal/agent may seek to memorize its reinforcement experience. To avoid capacity limits, it may seek to memorize a concise summary of this experience.

Value function

The value function is defined as the total reward expected after a state s, discounted over all future steps:

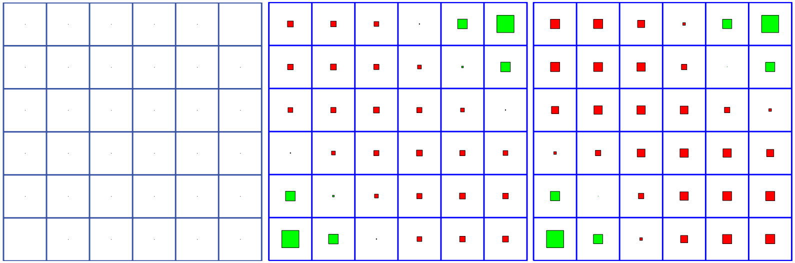
$$V_\pi(s) = \sum_a \pi(a,s) \left[r(a,s) + \gamma V_\pi(s') \right], \qquad s \xrightarrow{a} s'$$

Given the reinforcement structure, the value function can be computed by iteration:



Policy evaluation

Computation of the value function V is termed *policy evaluation*. This terminology emphasizes that the value function depends on policy!



Value functions looking ahead 0, 10, and 40 steps.

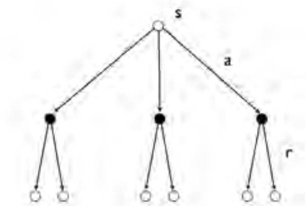
Optimal policy and value

A policy π is better than a policy π' if its expected return is larger for all states s and actions s:

$$V_\pi(s,a) \geq V_{\pi'}(s,a), \qquad \forall\ s,a$$

An optimal policy π^* is associated with an optimal value function:

$$V^*_{\pi^*}(s,a) = \max_\pi V_\pi(s,a), \qquad \forall\ s,a$$



Policy improvement

We now use the value function V_π to improve our policy π. Specifically, we adopt a probabilistic policy π' with the differential value of each action as its action value

$$m_a = V_\pi(s') - V_\pi(s), \qquad s \xrightarrow{a} s'$$

and the action probabilities

$$\pi'(a, s) = \frac{\exp(\beta\, m_a)}{\sum_{a'} \exp(\beta\, m_{a'})}$$

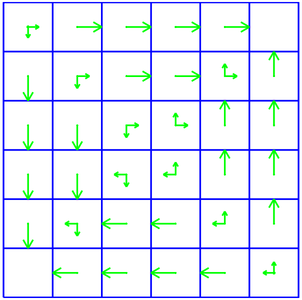
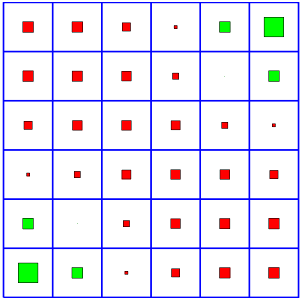
where the parameter β sets the "exploitation/exploration" balance.

Changing policy from π (random) to π' *invalidates* the value function V_π! Changing policy necessitates renewed "policy evaluation" (i.e., re-computing $V_{\pi'}$)!

Policy iteration

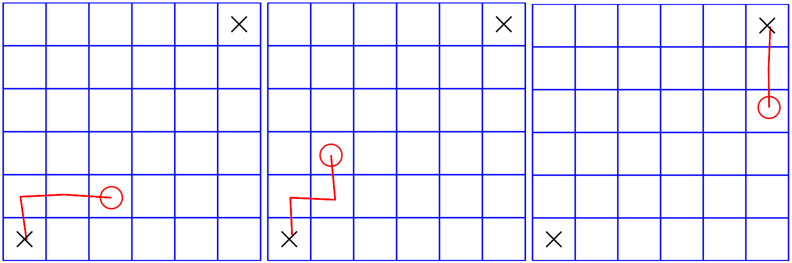
Iterating between *improving* the policy $\pi(a, s)$ and *evaluating* it by recomputing $V_\pi(s)$, we converge on the following (exploitative) policy:

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2$$

Improved policy

With its improved policy, the agent heads straight for the exit (provided the exploitation parameter β is sufficiently large):



Points to note

- A value function V_π may be used to improve a policy $\pi(s, a)$.
- *Policy evaluation:* the policy $\pi(s, a)$ is used to increase the value function $V_\pi(s)$:

$$V_\pi(s) = \sum_a \pi(a, s) \; \left[r(a, s) + \gamma V_\pi(s') \right]$$

- *Policy improvement:* the value function $V_\pi(s)$ is used to improve the policy $\pi(s, a)$:

$$m_a = V_\pi(s') - V_\pi(s) \qquad\qquad \pi(a, s) = \frac{\exp(\beta\, m_a)}{\sum_{a'} \exp(\beta\, m_{a'})}$$

- *Policy iteration:* both steps are carried out in alternation.

- All this is **mere theory** because the full reward structure $r(s, a)$ must be known!

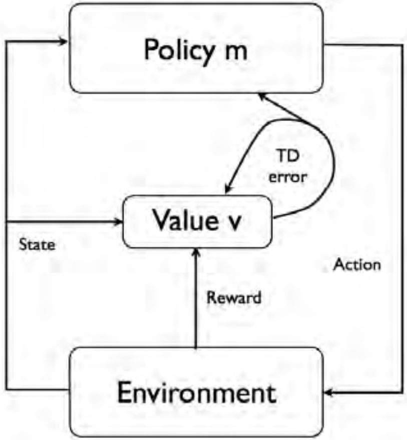
5 Gridworld with TD learning (plausible model)

Recall that TD learning is characterized by:

- Updating values at every time-step (instead of waiting until the entire reinforcement structure is known).
- Basing estimates on other estimates (bootstrapping).
- Deterministic convergence (difficult to prove).
- Maintaining separate value and policy functions.

Concurrent improvement

The TD prediction error can be used to concurrently improve both value function and policy.



ctd

Action and state transition:

$$a: \qquad s \;\rightarrow\; s'$$

TD prediction error (total expectation at s' and s may differ!)

$$\delta_a = r_{s'} + \gamma V_{s'} - V_s$$

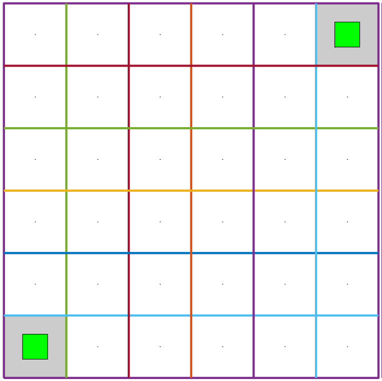
Policy evaluation:

$$V_s \rightarrow V_s + \alpha\,\delta_a$$

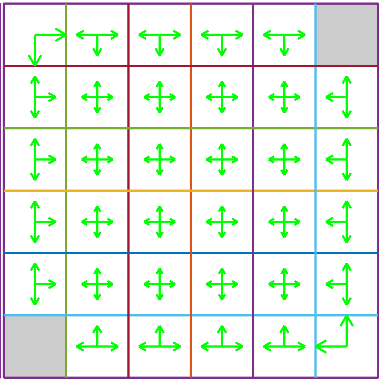
Policy improvement:

$$m_a \rightarrow m_a + \epsilon\,\delta_a$$

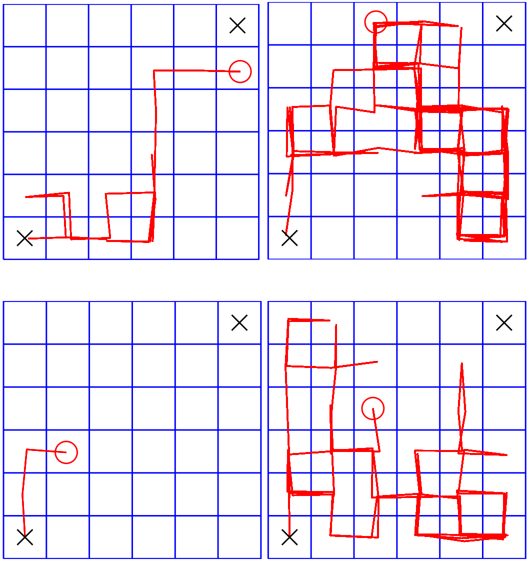
Initial state values



Initial action policy

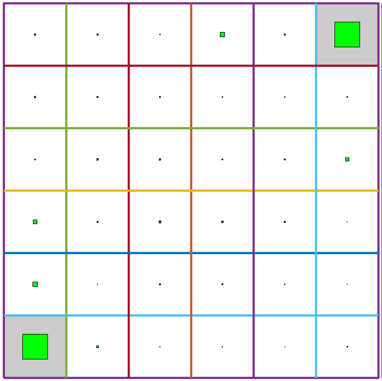


Example runs:

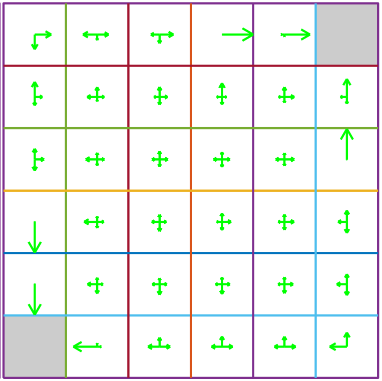


After 10 runs

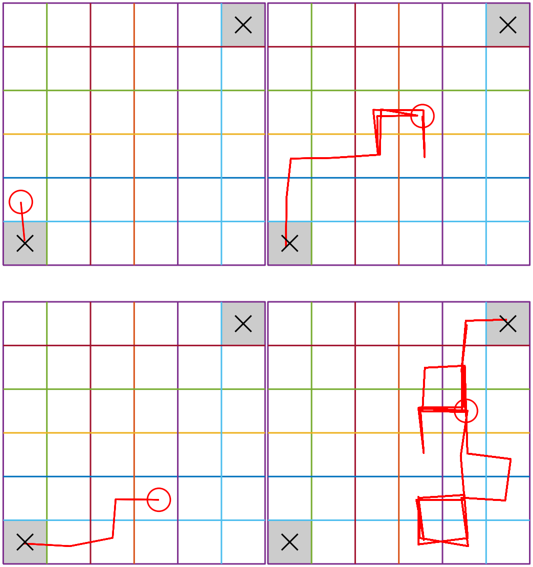
State values



Action policy

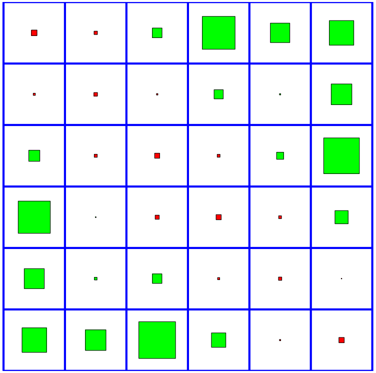


Example runs:

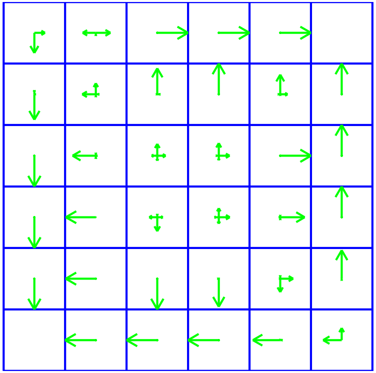


After 200 runs

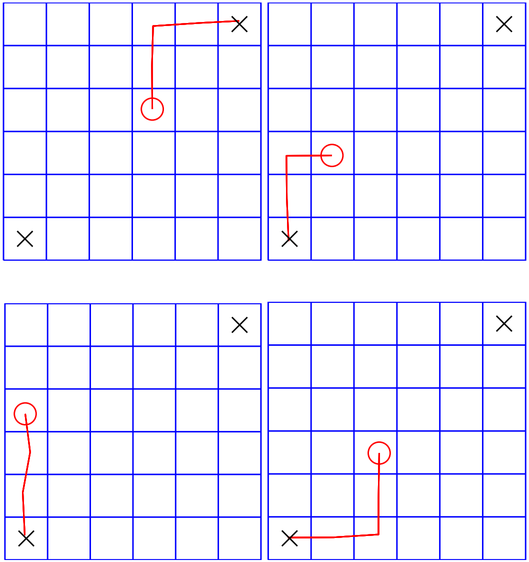
State values



Action policy

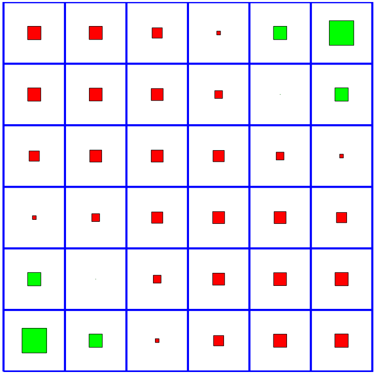


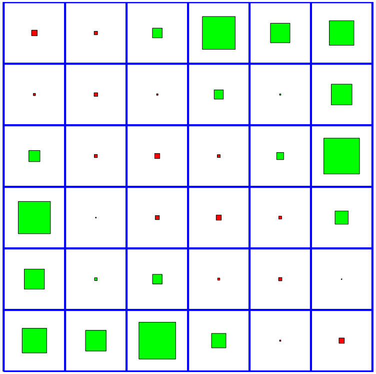
Example runs:



Recall value map for random policy

Compare value map for a random policy (from previous section):





What a difference policy makes: dumb (left) or smart (right)

Points to note

- *Policy evaluation* (learning expected reward) and *policy improvement* (learning action values) can be performed concurrently.
- Amazingly, this makes learning far more efficient than with *policy iteration*.
- The update rules are simple:

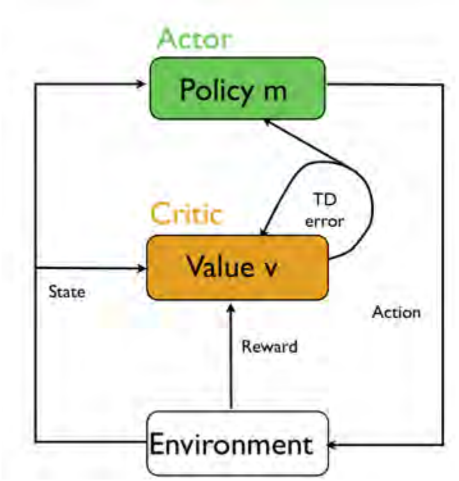
Policy evaluation	$V_s \rightarrow V_s + \alpha\,\delta_a$
Policy improvement	$m_a \rightarrow m_a + \epsilon\,\delta_a$

- TD prediction error is used in both:

$$\delta_a = r_{s'} + \gamma V_{s'} - V_s$$

Actor-critic models

- We have now met our first **actor-critic model**.
- Policy $\pi(a, s)$ is the *actor*, as it sets action preferences.
- Value function $V_\pi(s)$ is the *critic*, as it evaluates total reward expected from policy.



Next: Actor-critic models