RL in NL

A little reinforcement learning example in NetLogo

Nicolas Payette

SCHOOL OF GEOGRAPHY
AND THE ENVIRONMENT

UNIVERSITY OF
OXFORD

June 4th 2020

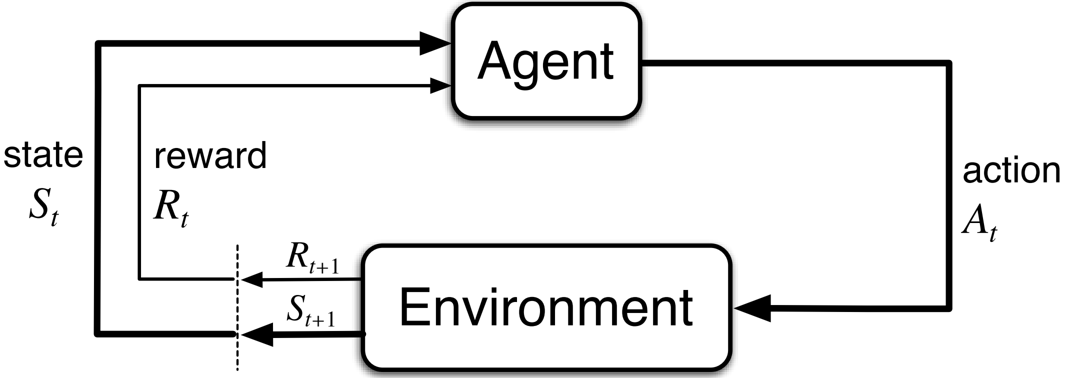
Use reinforcement learning when…

Agents’ **actions** generate **rewards** from the **environment** over time.

It can handle delayed reward: when actions taken **now** can affect **future** rewards.

But it’s trial-and-error search: agents have to try different actions to evaluate their effects.

Agent-environment interaction in RL



Sutton & Barto (2017) *Reinforcement Learning: An Introduction*

Temporal difference learning: TD(0)

Actions are chosen using an ϵ**-greedy** policy:

$$A \leftarrow \begin{cases} \text{a random action,} & \text{with probability } \epsilon \\ \text{the action that maximizes } V(S'), & \text{otherwise} \end{cases}$$

Agents must learn the value of the possible **states**:

$$V(S) \leftarrow V(S) + \alpha\left(\overbrace{R + \gamma V(S') - V(S)}^{\text{TD error}} \right)$$

Temporal difference learning: TD(0)

Actions are chosen using an ϵ**-greedy** policy:

$$A \leftarrow \begin{cases} \text{a random action,} & \text{with probability } \epsilon \\ \text{the action that maximizes } V(S'), & \text{otherwise} \end{cases}$$

Agents must learn the value of the possible **states**:

$$V(S) \leftarrow V(S) + \alpha\left(\overbrace{R + \gamma V(S') - V(S)}^{\text{TD error}} \right)$$

Constraint

Only works if the agent has a model of the environment: if it knows which action will bring about which state.

The Q function

$$Q : State \times Action \to Value$$

SARSA (on-policy – when you care about reward *while* learning):

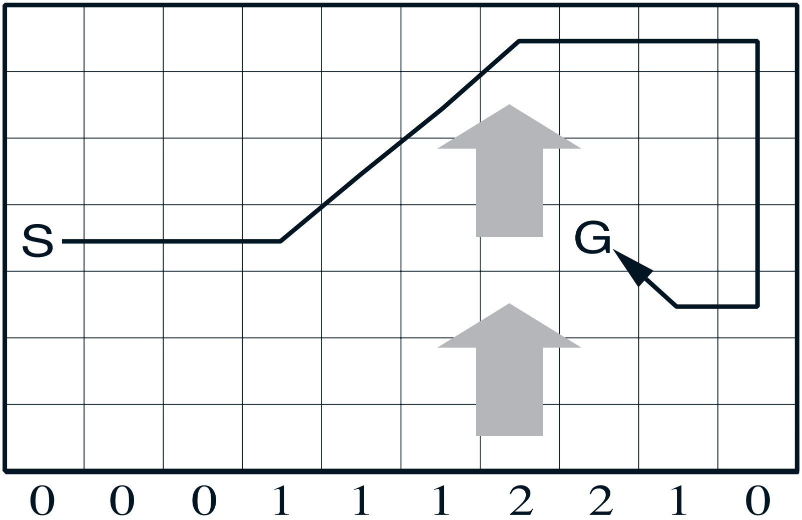
$$Q(S,A) \leftarrow \underbrace{Q(S,A)}_{\substack{\text{old}\\\text{value}}} + \underbrace{\alpha}_{\substack{\text{learning}\\\text{rate}}} \left(\overbrace{\underbrace{R}_{\text{reward}} + \underbrace{\gamma}_{\substack{\text{discount}\\\text{factor}}} \cdot \underbrace{Q(S',A')}_{\substack{\textbf{value of next}\\\textbf{action taken}}} - Q(S,A)}^{\text{TD error}} \right)$$

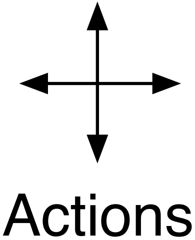
Q-Learning (off-policy – when you just care about finding the optimal policy):

$$Q(S,A) \leftarrow \underbrace{Q(S,A)}_{\substack{\text{old}\\\text{value}}} + \underbrace{\alpha}_{\substack{\text{learning}\\\text{rate}}} \left(\overbrace{\underbrace{R}_{\text{reward}} + \underbrace{\gamma}_{\substack{\text{discount}\\\text{factor}}} \cdot \underbrace{\max_a Q(S',a)}_{\substack{\textbf{value of best}\\\textbf{possible action}}} - Q(S,A)}^{\text{TD error}} \right)$$

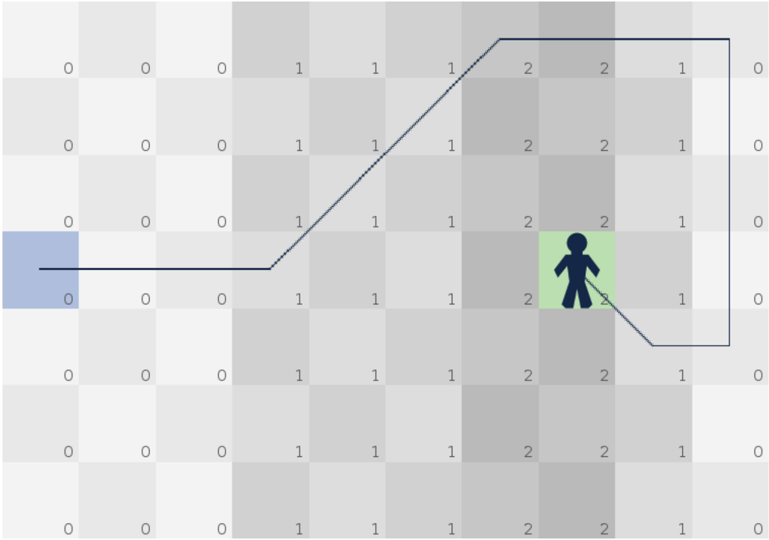
Still ϵ-greedy, but this time we choose the action with the best Q-value.

Example: the windy grid world





Sounds like a good candidate for a NetLogo model?



0	0	0	1	1	1	2	2	1	0
0	0	0	1	1	1	2	2	1	0
0	0	0	1	1	1	2	2	1	0
0	0	0	1	1	1	2	2	1	0
0	0	0	1	1	1	2	2	1	0
0	0	0	1	1	1	2	2	1	0
0	0	0	1	1	1	2	2	1	0