

Reinforcement Learning for Robots

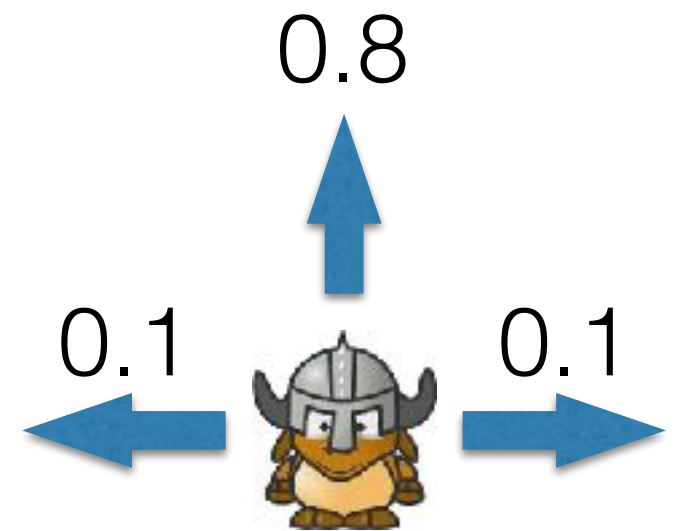
Michael Mistry

10/11/16

Treasure hunt game:



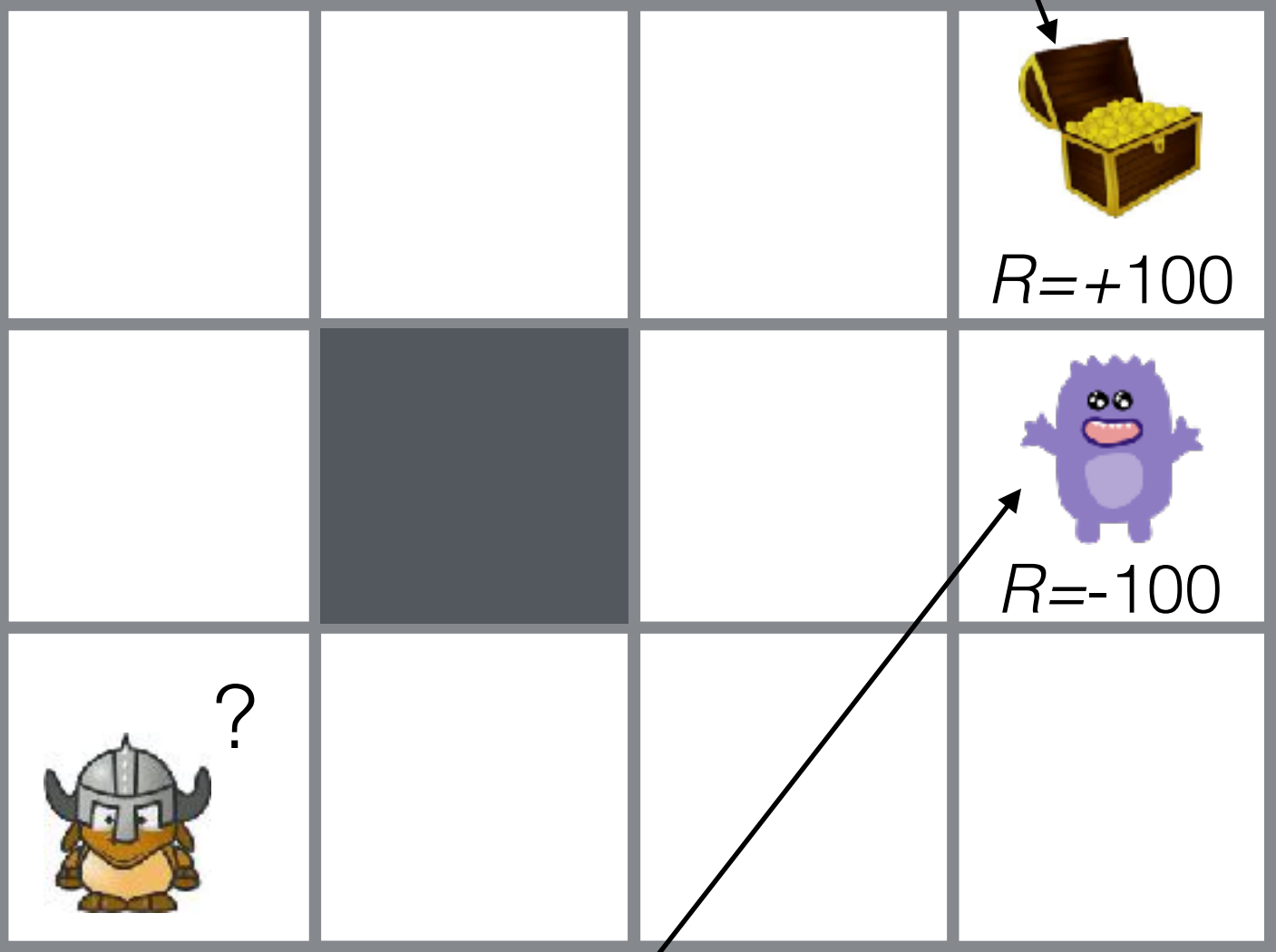
Rules of the game:



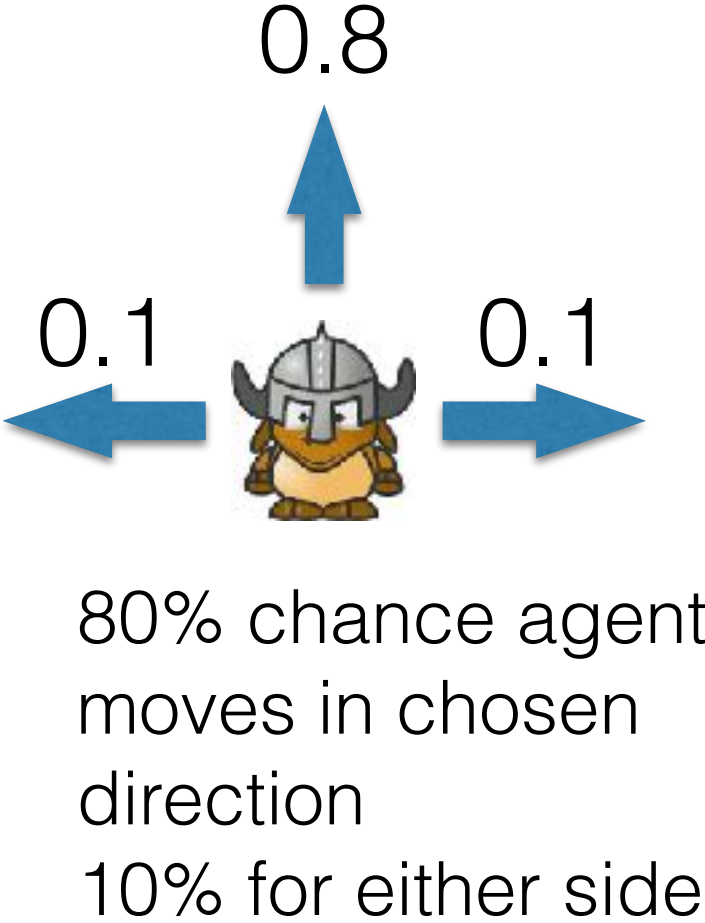
80% chance agent moves one space in its chosen direction, otherwise 10% chance (for each side) to move one space orthogonally

If agent hits a wall it stays where it is.

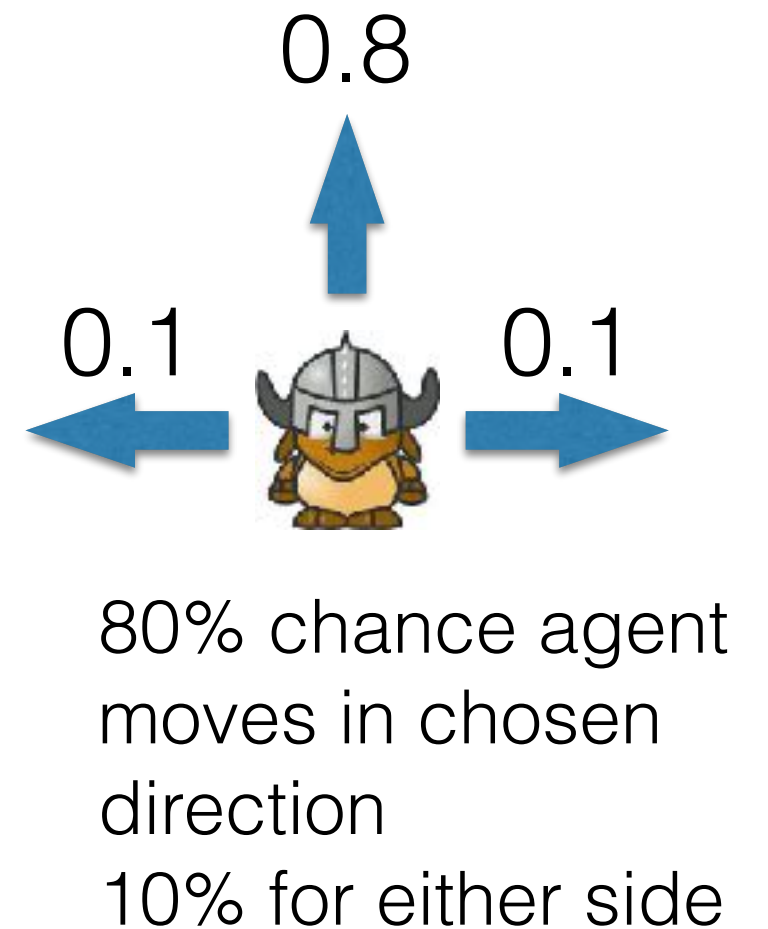
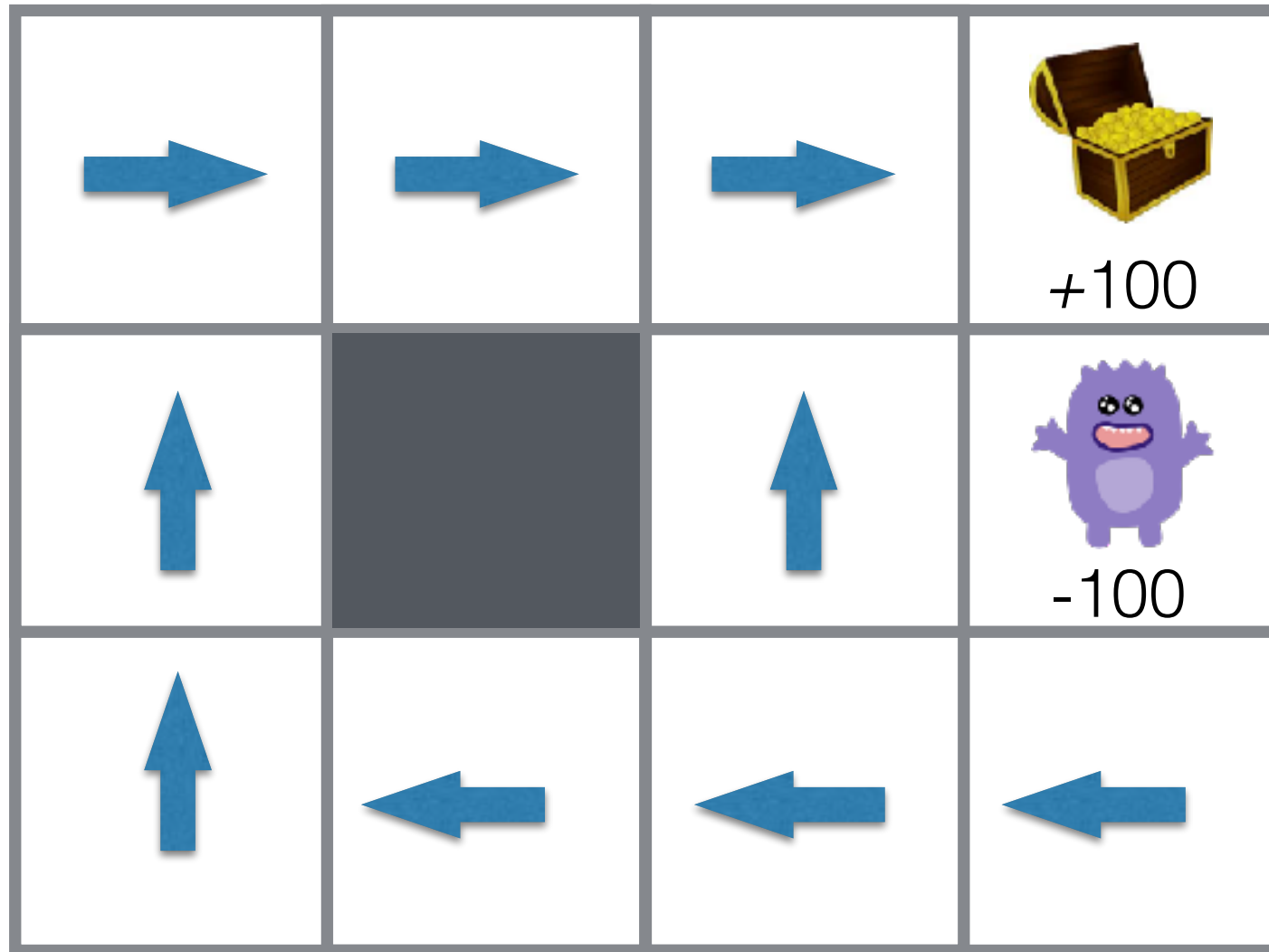
If agent gets to the gold, it gets a reward of 100 (and game ends)



If agent gets to the monster, it gets a reward of -100 (and game ends)



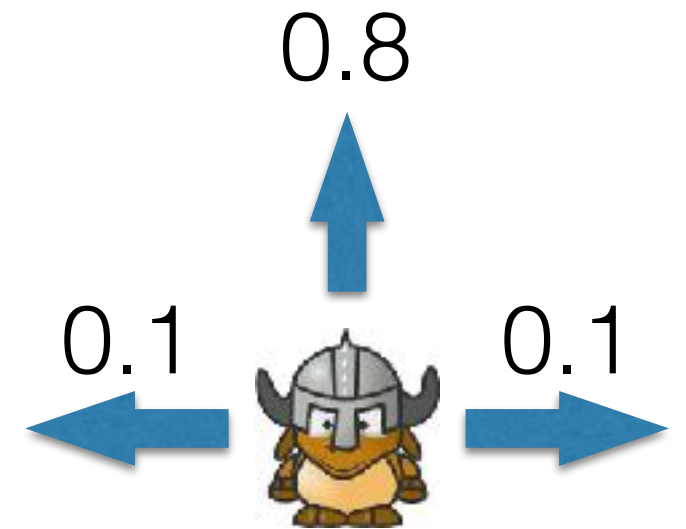
The following shows the optimal policy: $\pi^*(s)$
 if $R(s) = -4$ for the non-terminal states



A *policy* is a mapping from states to actions

An *optimal policy* is a policy that yields the highest possible expected utility

Assume the Agent knows nothing about the environment:
The Agent does not know where the gold or monster are (or even if there is gold or a monster), how big the maze is, where the walls are, etc.



80% chance agent moves in chosen direction
10% for either side

More specifically:

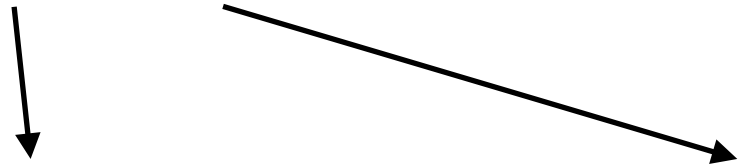
agent does not know transition probabilities: $P(s' | s, a)$

and does not know the reward function: $R(s)$

Utility Function (or value of each state)

$P(s'|s, a)$ and $R(s)$

are critical information for computing the Utility of each state:


$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a) U(s')$$

how valuable is the future (0 to 1)?

can we still infer, or *learn*, this Utility function, without prior knowledge of transition probabilities and the rewards?

Three main types of machine learning

Supervised Learning:

agent learns with a Teacher (learns with *labelled* data)

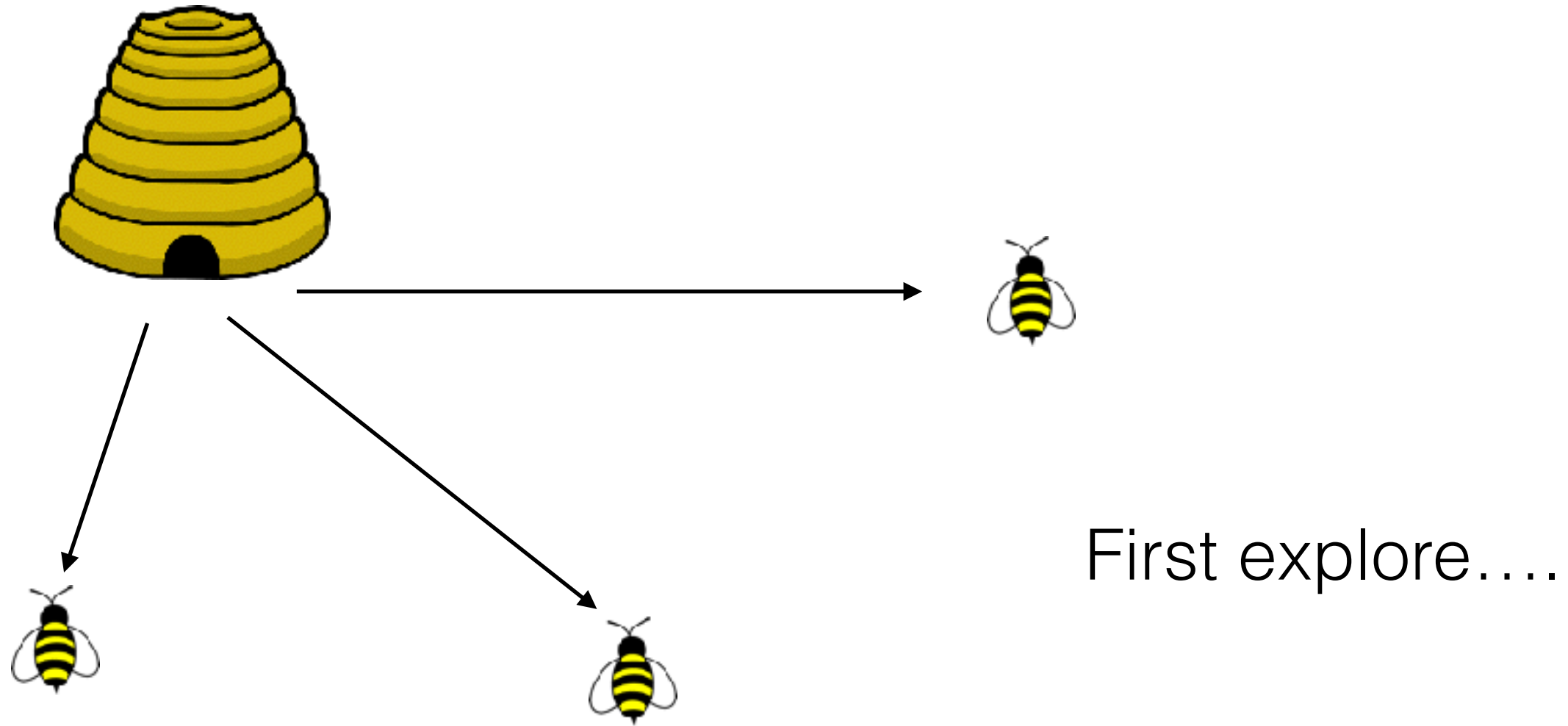
Unsupervised Learning:

agent must learn without a Teacher (unlabelled data),
find hidden patterns in data

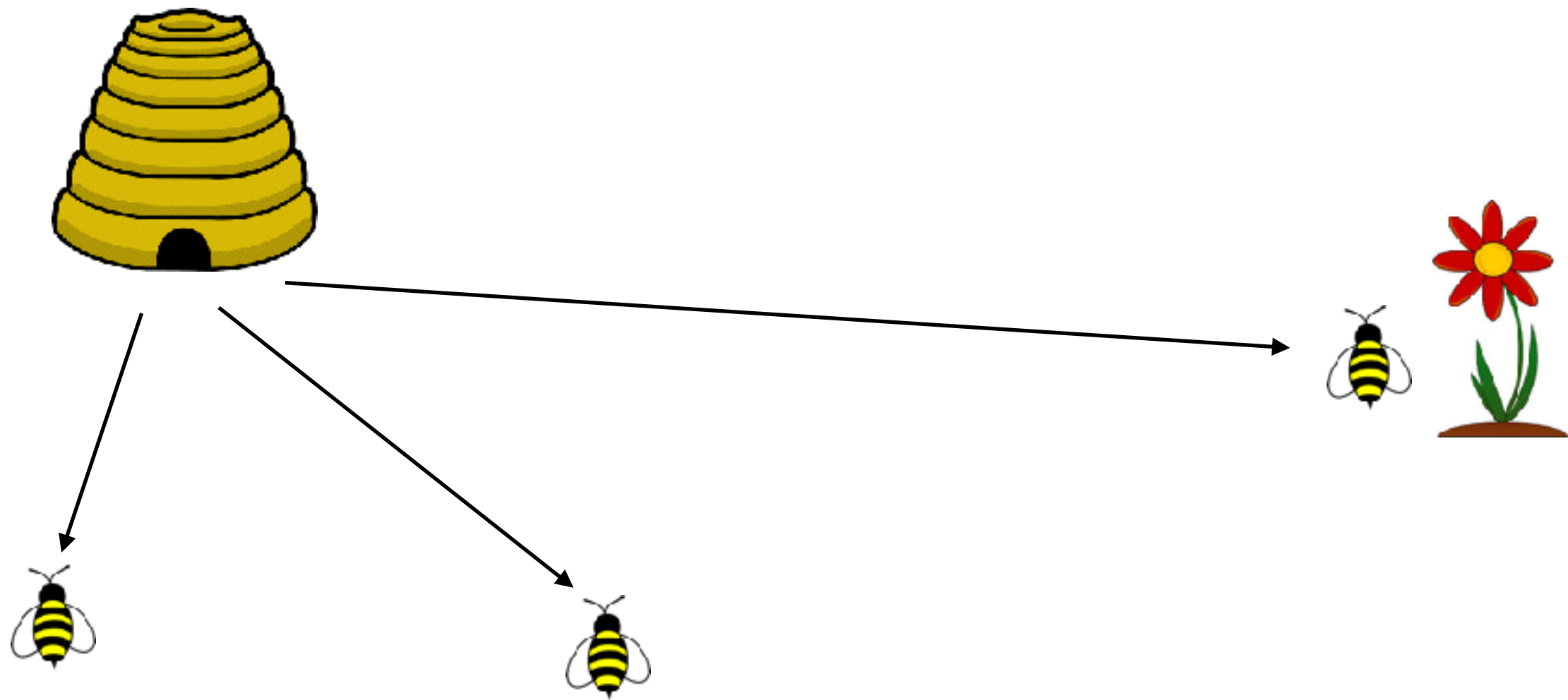
Reinforcement Learning:

agent's actions are rewarded or punished (only at certain times, not necessarily continuously) and then must learn the actions that maximise reward over time

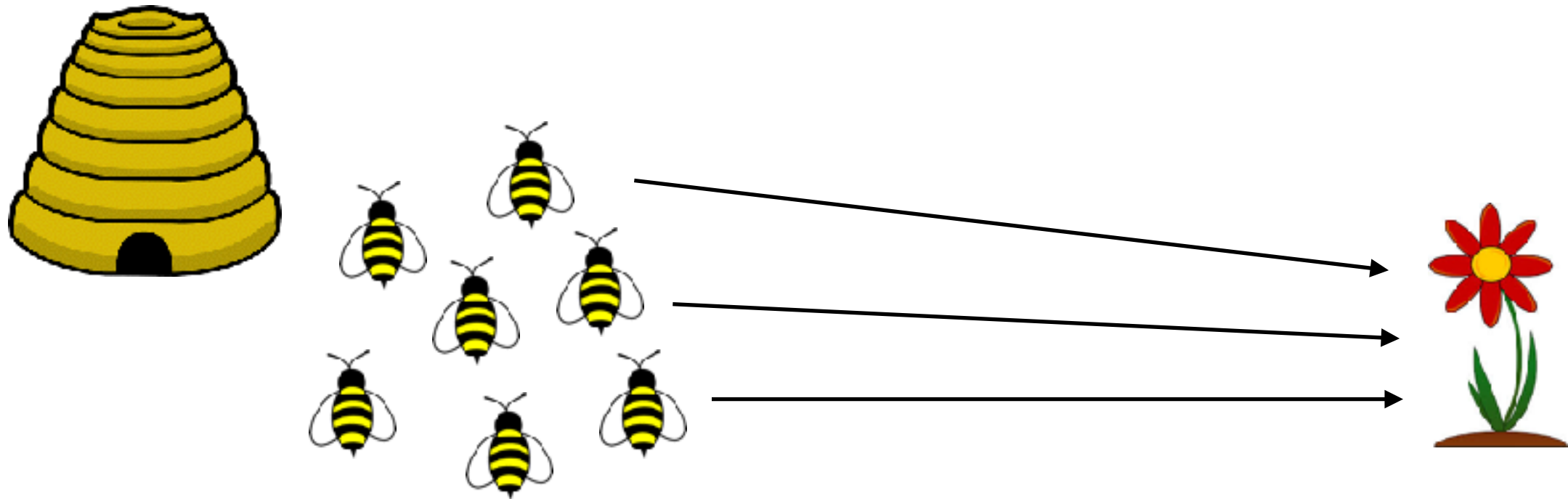
Reinforcement Learning: Agent must balance the trade off between *Exploration* and *Exploitation*



Reward Found!

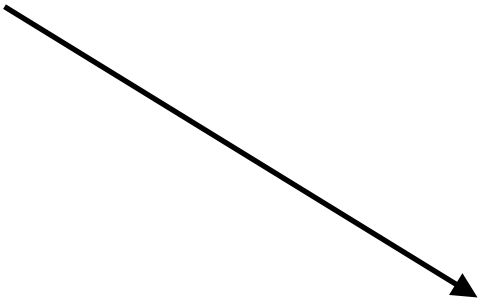
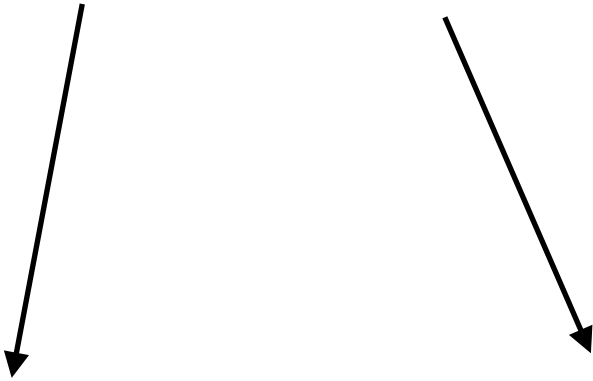
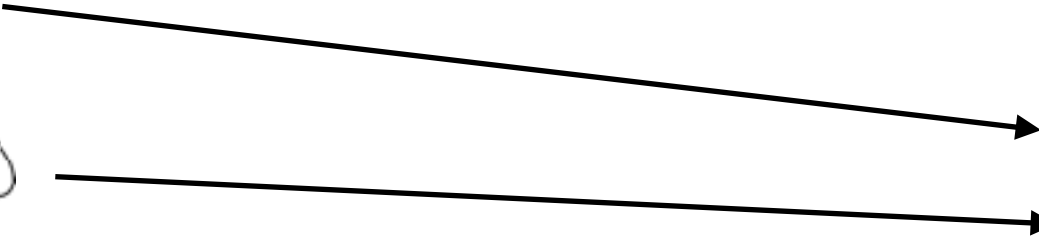


Now Exploit!



but there may be bigger rewards still
out there that we haven't found yet

Exploration vs. Exploitation: trade off between gaining reward now and searching for potential rewards in the future



Exploration: Trying new things (risky)

vs.


Exploitation: Going with what you know (safe)

Are you going to go to the same restaurant where you know you will get adequate “food”?



Or are you going to be adventurous and try a new Korean restaurant? (which may be amazing or awful)

The agent can update its utility function, from experience, using *Policy Iteration*:

$$U(\mathbf{s})_{i+1} = R(\mathbf{s}) + \gamma \sum_{\mathbf{s}'} P(\mathbf{s}' | \mathbf{s}, \pi(\mathbf{s})) U_i(\mathbf{s}')$$


use our *models* of the Reward
Function and Transition

fixed policy

Probabilities: estimates obtained
from experience

after sufficient experience with the fixed policy, we can try to update it, using the new estimate of $U(\mathbf{s})$:

$$\pi(\mathbf{s}) = \arg \max_a \sum_{\mathbf{s}'} P(\mathbf{s}' | \mathbf{s}, a) U(\mathbf{s})$$

Basic Reinforcement Learning Procedure:

1. Start with a Fixed Policy
2. Try the Policy: run a number of trials (rollouts) using the policy
3. Improve your model of the environment, based on the experience you collect in step 2
4. Update your policy and repeat from step 2

Encouraging Exploration (reward visiting unknown states)

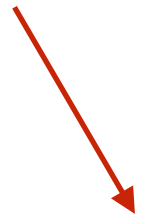
Instead of updating utilities as before:

$$U(s)_{i+1} = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a) U_i(s')$$

Use the following equation:

$$U^+(s)_{i+1} = R(s) + \gamma \max_a f \left(\sum_{s'} P(s'|s, a) U_i^+(s'), N(s, a) \right)$$

number of times action a has been tried in state s



with an exploration function f that depends on the number of experiences:

some fixed reward

$$f(u, n) = \begin{cases} R^+, & \text{if } n < N_e \\ u, & \text{otherwise} \end{cases}$$

number of times we want the agent to try an action in a state



Thus far we have discussed *model-based* Reinforcement Learning: we learn a model of the environment through experience. In particular we learn: $P(s'|s, a)$ and $R(s)$

However, transition policies can be difficult to estimate. How many parameters are required for $P(s'|s, a)$?

assuming n states, and m actions per state (and assuming each state may transition into any other state):

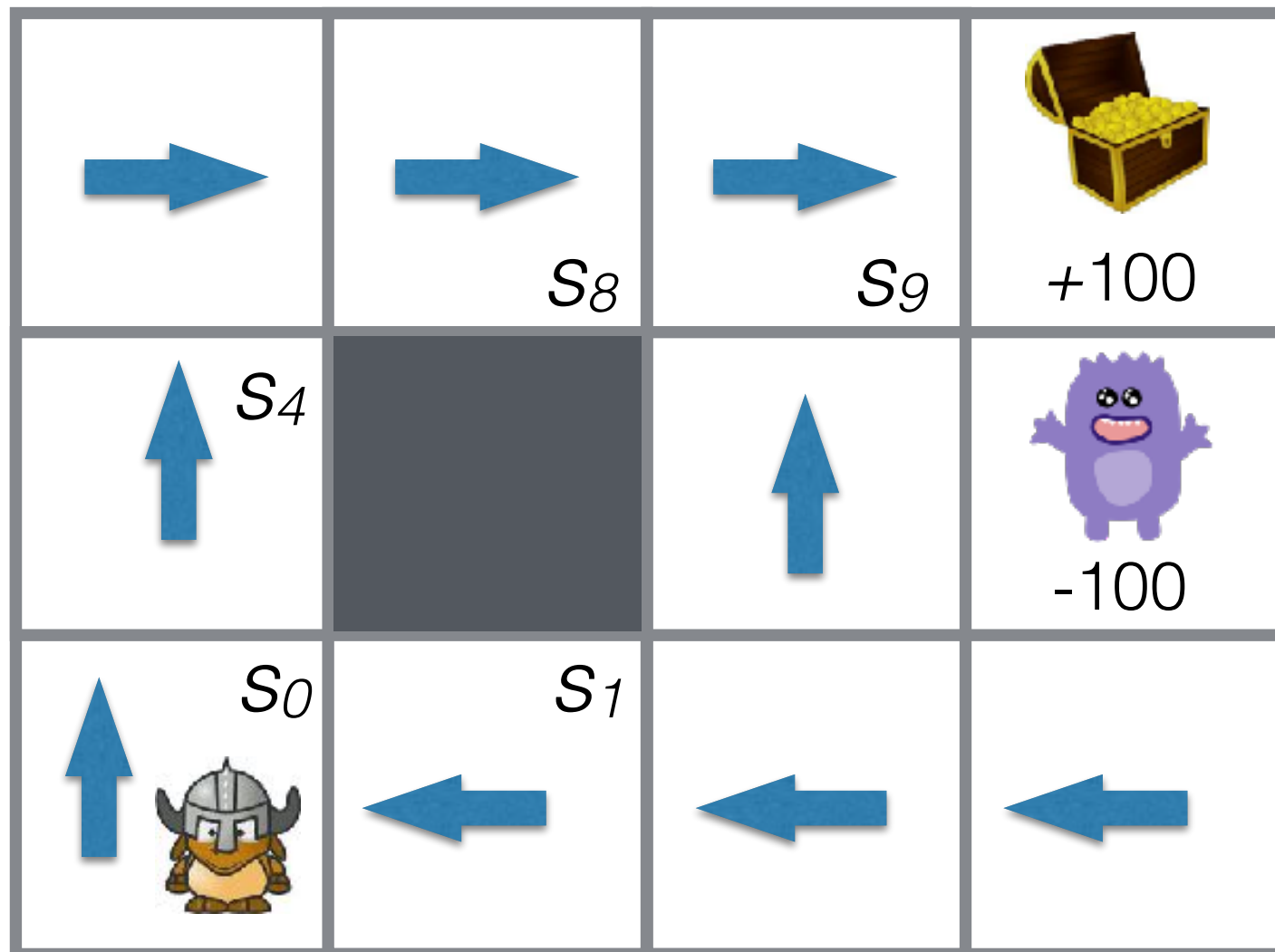
there are n^2m numbers to learn.

Also, smaller probabilities (rare occurrences) will be harder to learn (requiring many trials before experiencing it sufficiently)

Model-Free Reinforcement Learning

The agent tries to learn $U(s)$ without learning explicit models of transition probabilities

This can work because utilities of adjacent states are related in some way: *The utility experienced in a new state provides some information about the utility of the previous state.*



For example, with sufficient experience, the agent might infer:

$U(s_9)$ is approximately $\gamma 100 - 4$

$U(s_8)$ is approximately $\gamma U(s_9) - 4$

$U(s_0)$ is approximately $\gamma U(s_4) - 4$

This implies a learning rule:

whenever a transition occurs from s to s' , we update $U(s)$:

$$U_{i+1}(s) \approx R(s) + \gamma U_i(s')$$

This implies a general Temporal Difference learning rule:

whenever a transition occurs from s to s' , we update $U(s)$:

$$U_{i+1}(s) = U_i(s) + \alpha(R(s) + \gamma U_i(s') - U_i(s))$$



learning rate: a small number (<1) that controls how fast $U(s)$ changes in each iteration

$U(s)$ is slightly updated, based on change in utility to the next state

no need to learn transition probabilities!

We can learn the Utility function $U(s)$, without transition probabilities, however we still need them to get the policy??

$$\pi(s) = \arg \max_a \sum_{s'} P(s'|s, a) U(s)$$

$$U_{i+1}(s) = U_i(s) + \alpha(R(s) + \gamma U_i(s') - U_i(s))$$

Instead of learning $U(s)$, we can learn the Utility of a (state,action) pair:

$$Q_{i+1}(s, a) = Q_i(s, a) + \alpha(R(s) + \gamma \max_{a'} Q_i(s', a') - Q_i(s, a))$$

$Q(s, a)$ is the expected utility of taking action a in state s

Policy then becomes:

$$\pi(s) = \arg \max_a Q(s, a)$$

again no need to learn the transition probability!

This is called *Q-Learning*

Some problems with RL in Robotics:

Robotic systems are high-dimensional with continuous states and actions. (e.g. Q-Learning quickly becomes intractable)

The state is typically partially-observable and noisy (difficult to estimate true state)

Gaining experience can be tedious, expensive, dangerous (e.g. can damage robot, think about learning to walk!)

Initial conditions (starting over in same state) are difficult to reproduce.

Some methods may use simulation to aid these problems, but simulation is never a substitute for real experience.

Specifying a good reward function is challenging and may require a fair amount of domain knowledge.

To cope with these problems in robotics, we often use *Policy Search* methods:

Start with a reasonable policy (e.g. one you know won't break the robot), and improve it with each trial:

$$\theta_{i+1} = \theta_i + \alpha \nabla_{\theta} J$$

policy parameters



gradient of
expected return




We can estimate the gradient via least squares:

For p episodes, and a reference policy θ_i :

$$\nabla_{\theta} J \approx (\Delta \Theta^T \Delta \Theta)^{-1} \Delta \Theta^T \Delta \hat{J}$$

Stack of p
perturbations: $\Delta \theta_p$



Stack of p influences on
the return:



$$\Delta \hat{J}_p \approx J(\theta_i + \Delta \theta_p) - J(\theta_i)$$

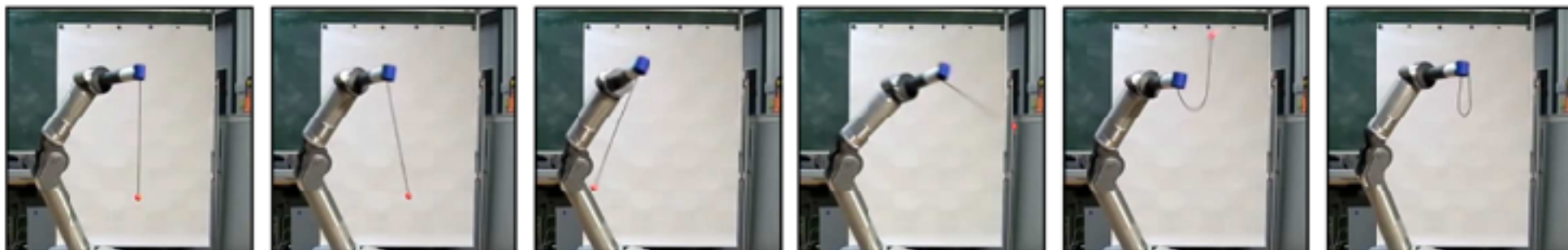
Learning Ball in Cup:



(a) Schematic drawings of the ball-in-a-cup motion

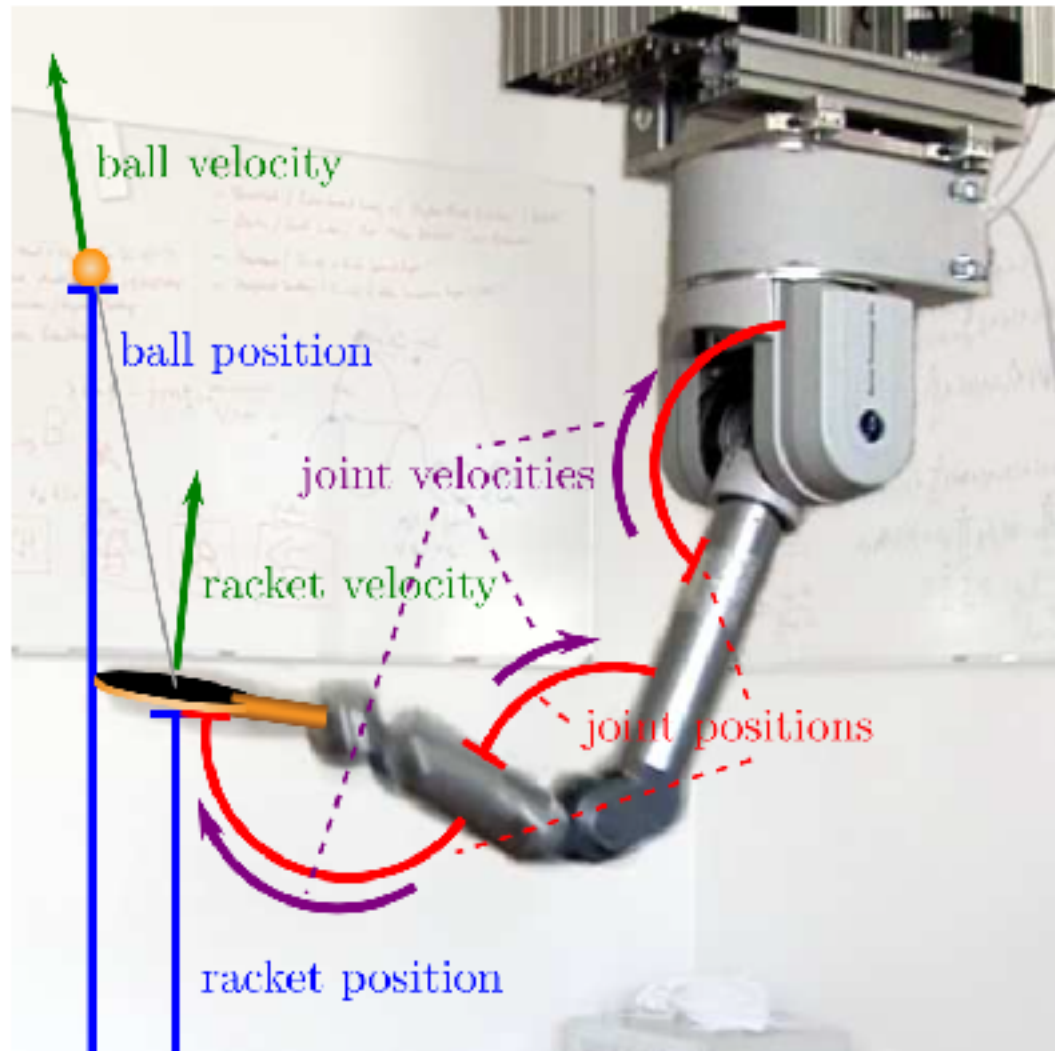


(b) Kinesthetic teach-in



(c) Final learned robot motion

Robot's state has 20 dimensions:





7 joints, 3 for ball, plus velocities for each

action is 7 dim (torque at each joint)

Reward Function:

cup position ball position

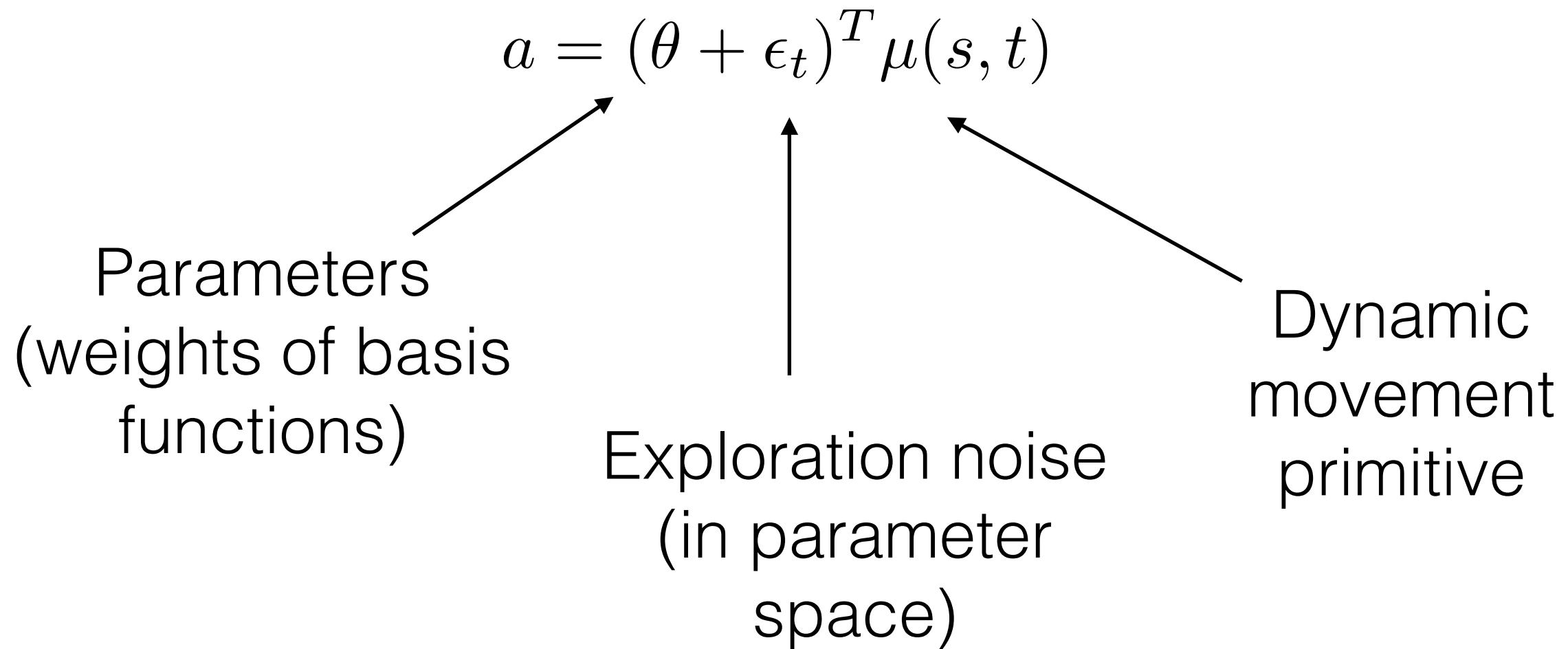

$$r(t_c) = \exp(-\alpha(x_c - x_b)^2 - \alpha(y_c - y_b)^2)$$


time when ball crosses rim *downward*

$$r(t) = 0 \text{ otherwise}$$

This reward function already encodes much domain knowledge about the task (i.e. it would be significantly more difficult and time consuming if the robot needed to learn that the ball must move downward into cup)

Policies are encoded as *parameterised* motor primitives:



Parameter Update Rule (PoWER):

Policy learning by Weighting Exploration with the Returns (PoWER) algorithm. PoWER is an expectation-maximization inspired algorithm that employs state-dependent exploration (as discussed in Section 7.2). The update rule is given by

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + E \left\{ \sum_{t=1}^T \mathbf{W}(\mathbf{s}_t, t) Q^\pi(\mathbf{s}_t, \mathbf{a}_t, t) \right\}^{-1} E \left\{ \sum_{t=1}^T \mathbf{W}(\mathbf{s}_t, t) \epsilon_t Q^\pi(\mathbf{s}_t, \mathbf{a}_t, t) \right\},$$

where $\mathbf{W}(\mathbf{s}_t, t) = \boldsymbol{\mu}(\mathbf{s}, t) \boldsymbol{\mu}(\mathbf{s}, t)^\top \left(\boldsymbol{\mu}(\mathbf{s}, t)^\top \hat{\boldsymbol{\Sigma}} \boldsymbol{\mu}(\mathbf{s}, t) \right)^{-1}$. Intuitively, this update can be seen as a reward-weighted imitation, (or recombination) of previously seen episodes.

essentially a more clever way to compute the policy gradient

Reference:

Kober, Jens, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey." *The International Journal of Robotics Research* (2013)

Videos of ball-in-cup (and some other tasks):

<http://www.dcsc.tudelft.nl/~jkober/research.htm>

Pancake flipping:

https://www.youtube.com/watch?v=W_gxLKSsSIE