Basics of Reinforcement Learning

Overview

HMMs The probability transformation matrix A allows forward and backward propagation

Dynamic programing An objective function and *dynamics* uses backward propagation

Reinforcement Learning A reward function and probabilistic dynamics uses forward propagation with iteration

Model-free Reinforcement Learning A reward function and probabilistic dynamics uses forward propagation with iteration and model estimation

Dynamic Programing

Final value problem

The dynamics is expressed by a difference equation,

$$\mathbf{x}(k+1) = f[\mathbf{x}(k), \mathbf{u}(k)]$$

The initial condition is:

$$\mathbf{x}(0) = \mathbf{x}_0$$

The allowable control is also discrete:

$$\mathbf{u}(k) \in U, k = 0, \ldots, N$$

The integral in the objective function is expressed as a sum:

$$J = \psi[\mathbf{x}(T)] + \sum_{0}^{N-1} \ell[\mathbf{u}(k), \mathbf{x}(k)]$$

It's easy to see that at the end,

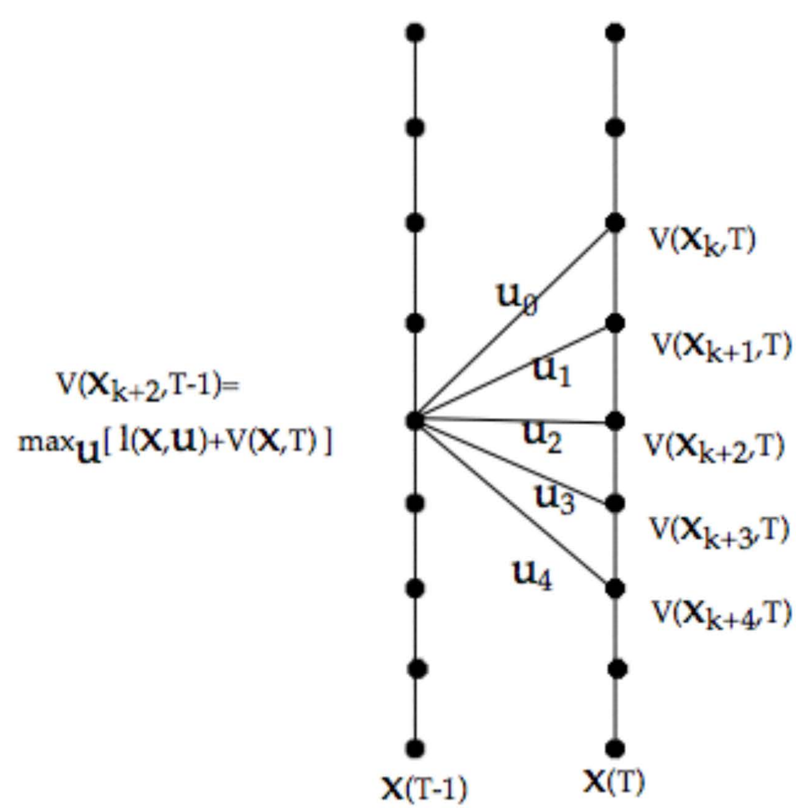
$$V(\mathbf{x}, N) = \psi[\mathbf{x}(N)]$$

One step back,

$$V(\mathbf{x}, N-1) = \max_{\mathbf{u} \in U} \{\ell[\mathbf{u}(N-1), \mathbf{x}(N-1)] + \psi[\mathbf{x}(N)]\}$$

And in general, for $k < N - 1$,

$$V(\mathbf{x}, k-1) = \max_{\mathbf{u} \in U} \{\ell[\mathbf{u}(k-1), \mathbf{x}(k-1)] + V[\mathbf{x}(k), k]\}, k = 0, \ldots, N$$



Overview

HMMs The probability transformation matrix A
allows forward and backward propagation

Dynamic programing An objective function and *dynamics*
uses backward propagation

RL Definitions

An MDP consists of a 4-tuple (S, A, T, R)

S being the set of possible states,

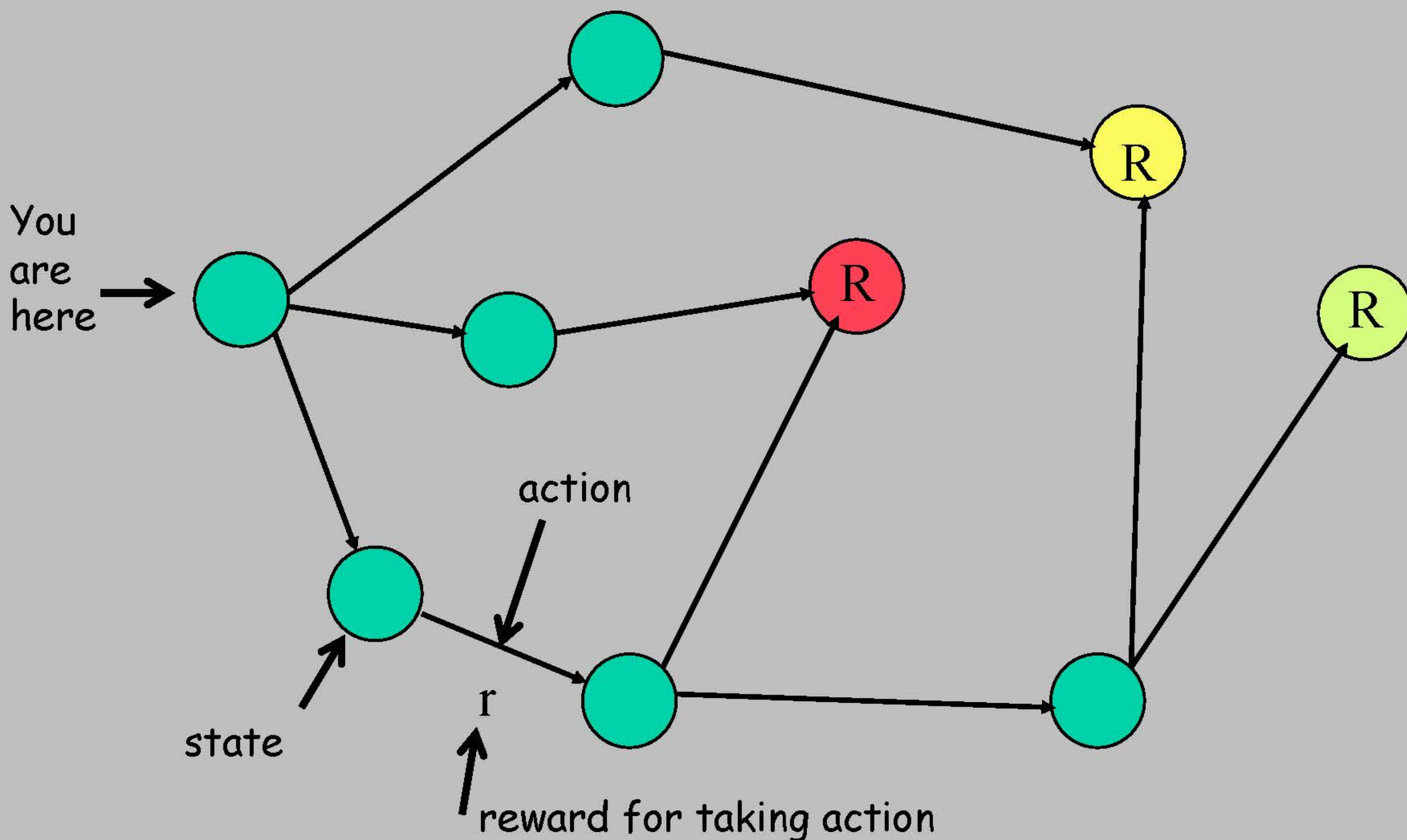
A the set of possible actions,

T the transition model describing the probabilities

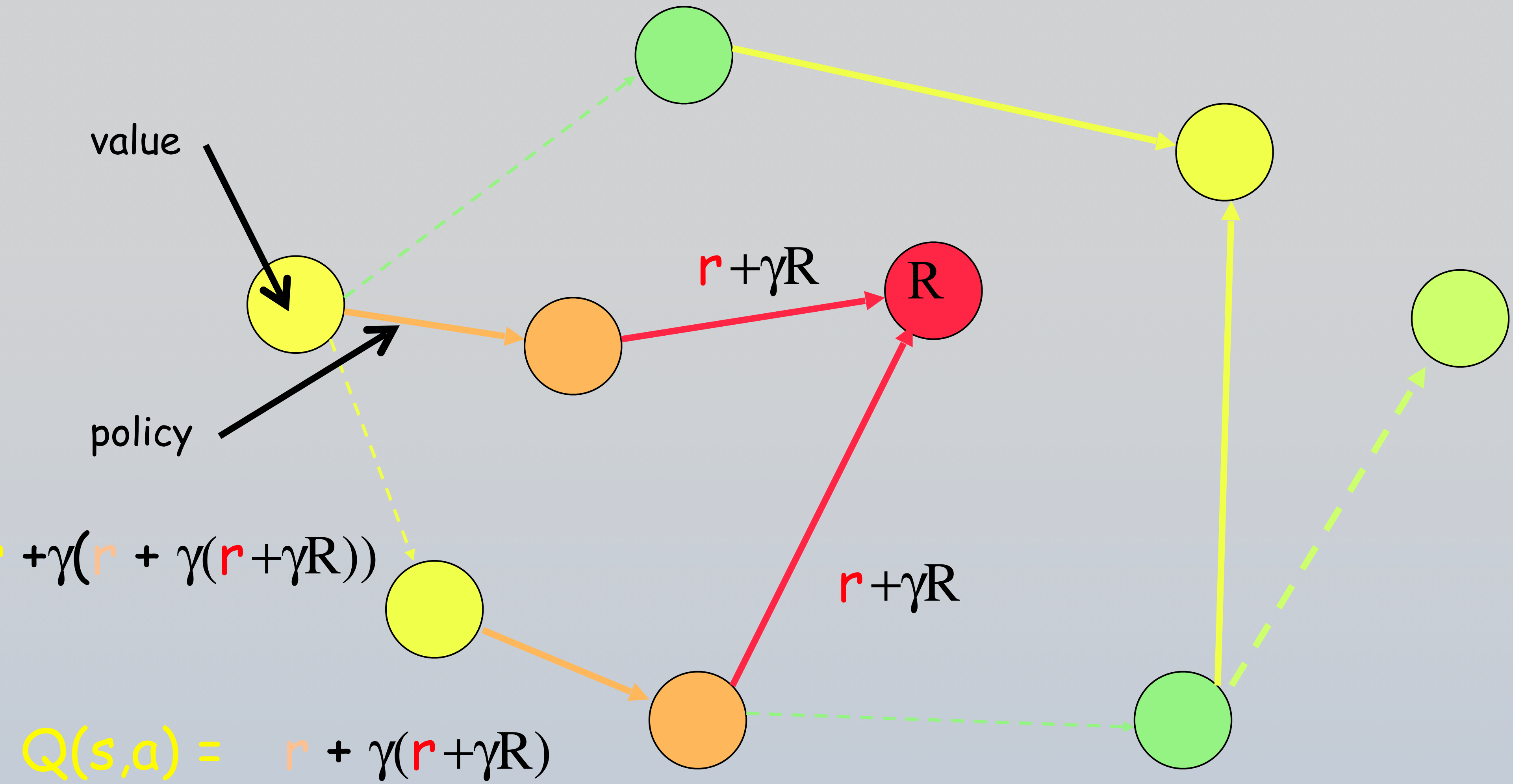
$P(s_{t+1}|s_t, a_t)$ of reaching a state s_{t+1} from state s_t at time t and executing action a_t,

R is the expected value of the reward r_t, distributed according to $P(r_t|s_t, a_t)$ and is associated with the transition from state s_t to some state s_{t+1} when executing action a_t.

Reinforcement Learning Primer : Before Learning



Reinforcement Learning Primer



By trying different actions from different starting points, gradually learn the expected reward value from any starting point

The basic RL algorithm w Model

We will speak of the optimal *value* of a state--it is the expected infinite discounted sum of reward that the agent will gain if it starts in that state and executes the optimal policy. Using π as a complete decision policy, it is written

$$V^*(s) = \max_{\pi} E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) \quad .$$

This optimal value function is unique and can be defined as the solution to the simultaneous equations

$$V^*(s) = \max_{a}\left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')\right), \forall s \in S \quad , \qquad (1)$$

which assert that the value of a state *s* is the expected instantaneous reward plus the expected discounted value of the next state, using the best available action. Given the optimal value function, we can specify the optimal policy as

$$\pi^*(s) = \arg\max_{a}\left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')\right) \quad .$$

Value Iteration

```
initialize V(s) arbitrarily

loop until policy good enough

        loop for  s ∈ S

                loop for  a ∈ A

                        Q(s,a) := R(s,a) + γ Σ_{s'∈S} T(s,a,s')V(s')

                V(s) := max_a Q(s,a)

        end loop

end loop
```

Policy Iteration

choose an arbitrary policy π'

loop

$\pi := \pi'$

compute the value function of policy π :

solve the linear equations

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$$

improve the policy at each state:

$$\pi'(s) := \arg\max_a \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s') \right)$$

until $\pi = \pi'$

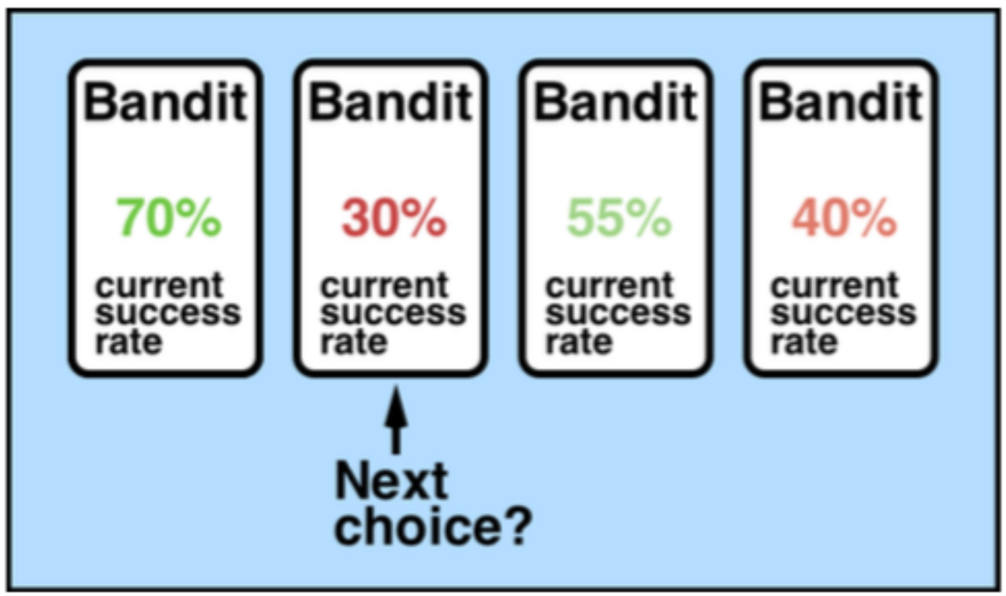
Overview

HMMs The probability transformation matrix A
allows forward and backward propagation

Dynamic programing An objective function and *dynamics*
uses backward propagation

Reinforcement Learning A reward function and
probabilistic dynamics uses forward propagation
Solution methods:
Value Iteration
Policy Iteration

The multi-armed bandit problem is a classic reinforcement learning example where we are given a slot machine with n arms (bandits) with each arm having its own rigged probability distribution of success. Pulling any one of the arms gives you a stochastic reward of either $R=+1$ for success, or $R=0$ for failure. Our objective is to pull the arms one-by-one in sequence such that we maximize our total reward collected in the long run.



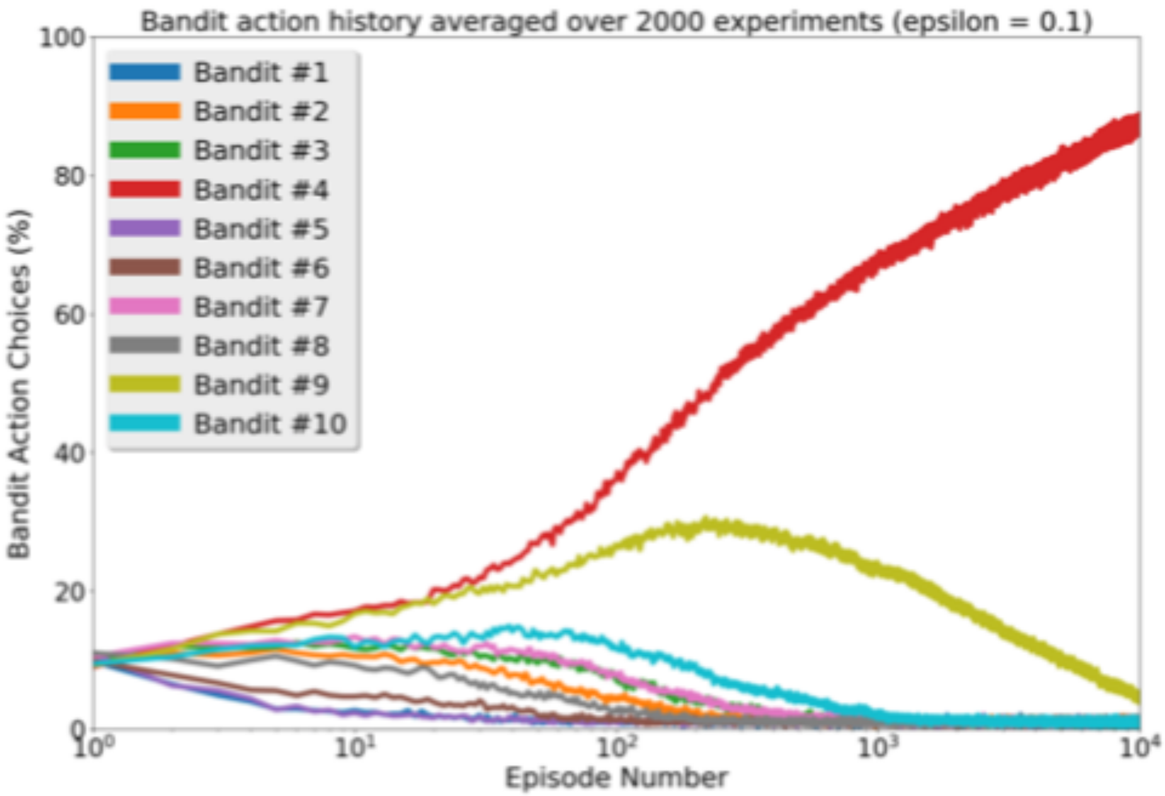


Fig 1) Bandit choices by the epsilon-greedy agent (epsilon = 10%) throughout its training

$$Q_k(a) = \frac{1}{k}(r_1 + r_2 + \ldots + r_k)$$

$$Q_{k+1}(a) = Q_k(a) + \frac{1}{k+1}(r_{k+1} - Q_k(a))$$

$$a_{greedy} = \underset{a}{\mathrm{argmax}} Q_k(a)$$

Temporal Difference Learning

$$\bar{V}_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where $0 \leq \gamma < 1$. This formula can be expanded

$$\bar{V}_t = r_t + \sum_{i=1}^{\infty} \gamma^i r_{t+i}$$

by changing the index of i to start from 0.

$$\bar{V}_t = r_t + \sum_{i=0}^{\infty} \gamma^{i+1} r_{t+i+1}$$

$$\bar{V}_t = r_t + \gamma \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}$$

$$\bar{V}_t = r_t + \gamma \bar{V}_{t+1}$$

Thus, the reinforcement is the difference between the ideal prediction and the current prediction.

$$r_t = \bar{V}_t - \gamma \bar{V}_{t+1}$$

Q-Learning variant of Temporal Difference Learning

The goal of RL is to find a policy π that maps from the set of states S to actions A so as to maximize the expected total discounted future reward

$$V^{\pi}(s) = E^{\pi}\Big(\sum_{t=0}^{\infty} \gamma^t r_t\Big) \tag{1}$$

Alternatively, the values can be parametrized by state and action pairs, denoted by $Q^{\pi}(s,a)$.

$$Q^*(s,a) = \sum_r r P(r|s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} Q^*(s',a') \tag{2}$$

Temporal difference learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \delta_Q \tag{3}$$

$$\delta_Q = r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t). \tag{4}$$

Overview

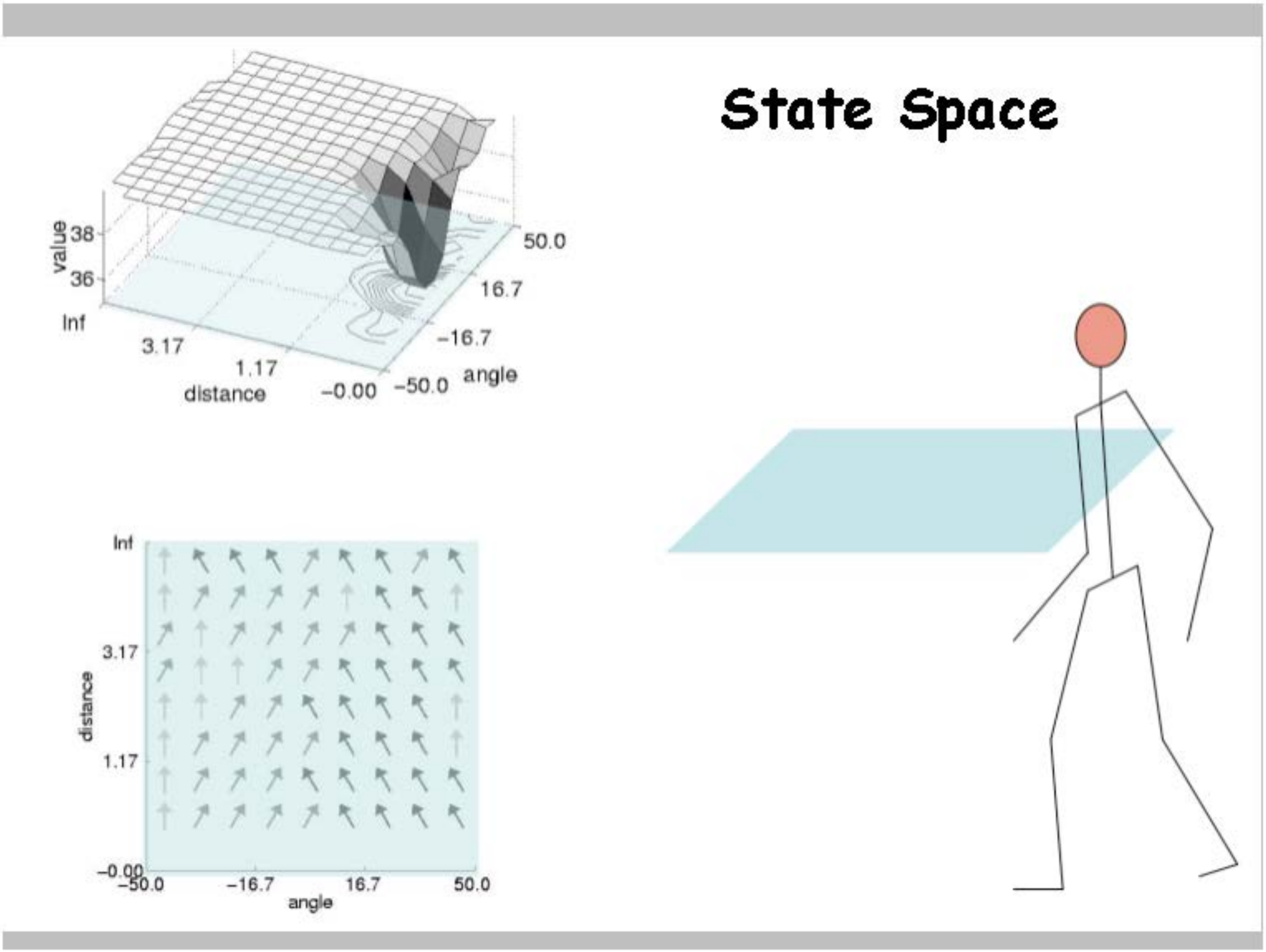
HMMs The probability transformation matrix A
allows forward and backward propagation

Dynamic programing An objective function and *dynamics*
uses backward propagation

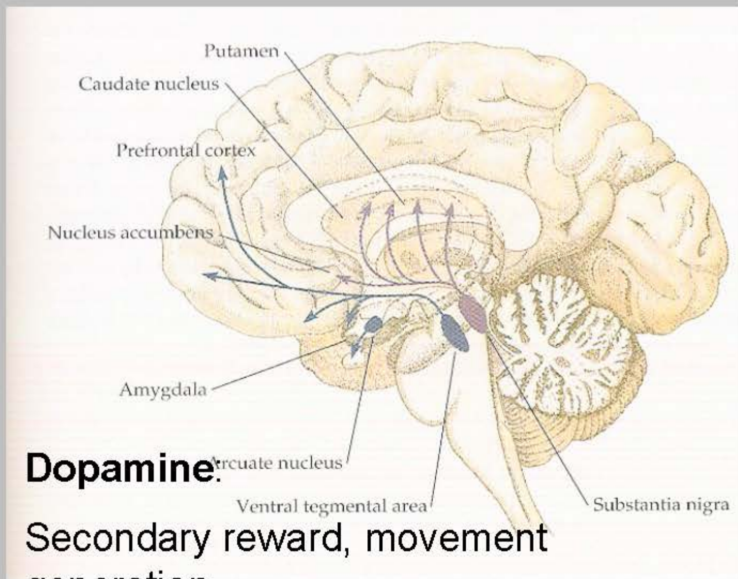
Reinforcement Learning A reward function and
probabilistic dynamics uses forward propagation
Solution methods:
Value Iteration
Policy Iteration

Model-free Reinforcement Learning A reward function and
probabilistic dynamics uses forward propagation with
iteration and *online model estimation*

Avoiding obstacles while walking

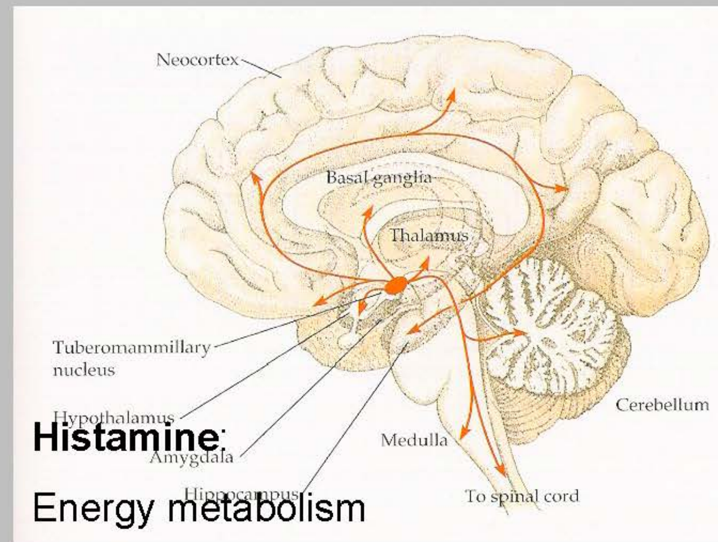


Dopamine: the brain's reward signal



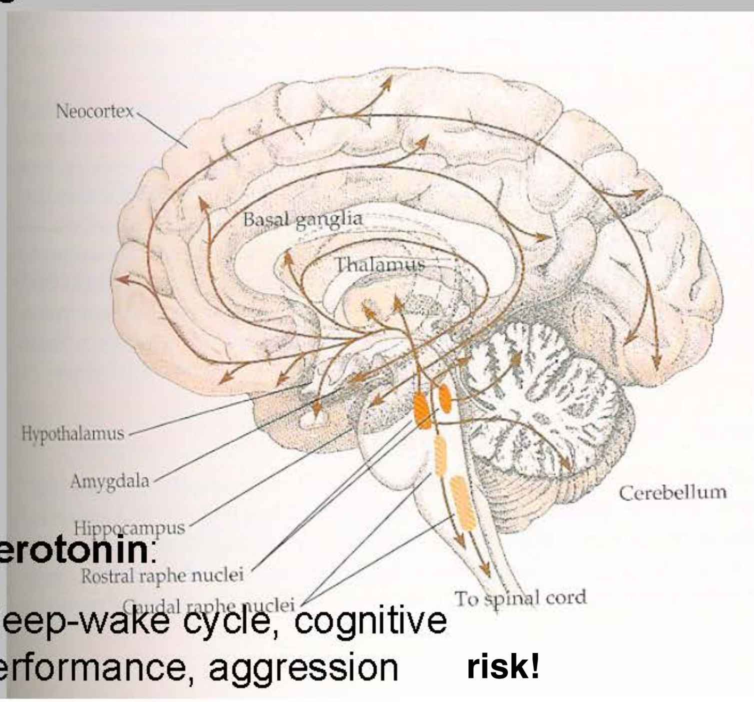
Dopamine:

Secondary reward, movement generation



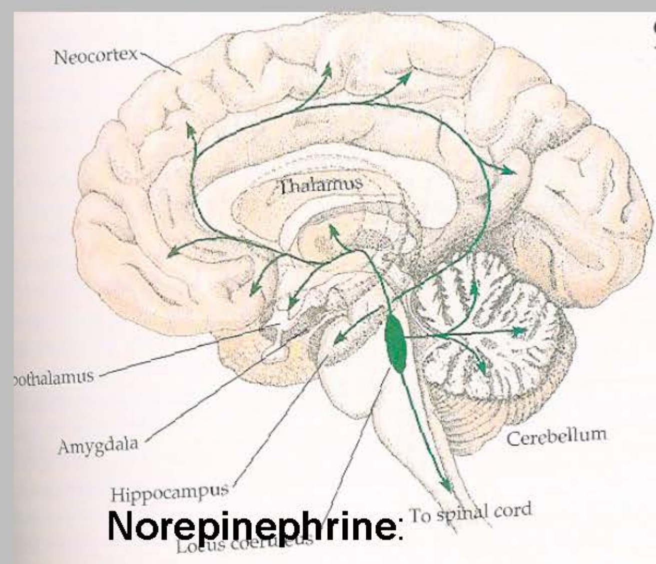
Histamine:

Energy metabolism



Serotonin:

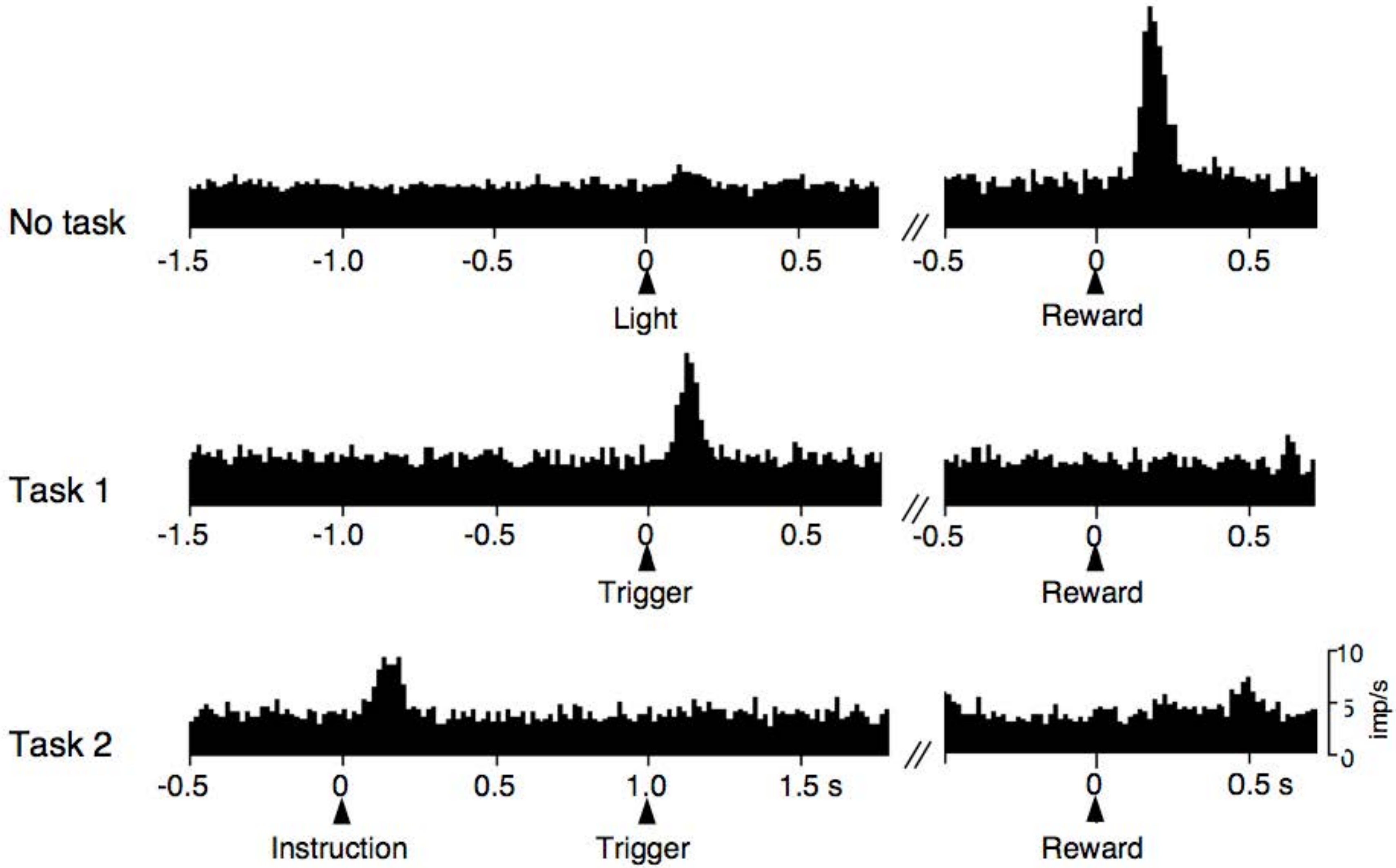
Sleep-wake cycle, cognitive performance, aggression **risk!**



Norepinephrine:

Attention, arousal, circadian rhythms

A Monkey uses Secondary Reward



Map of Temporal Discounting
(Tanaka et al., 2004 (Kenji Doya))

- Markov decision task with delayed rewards
- Regression by values and TD errors
  - with different discounting factors g