Topic 13: Introduction to Reinforcement Learning

Xiaolin Hu
Dept. of Computer Science and Technology
Tsinghua University

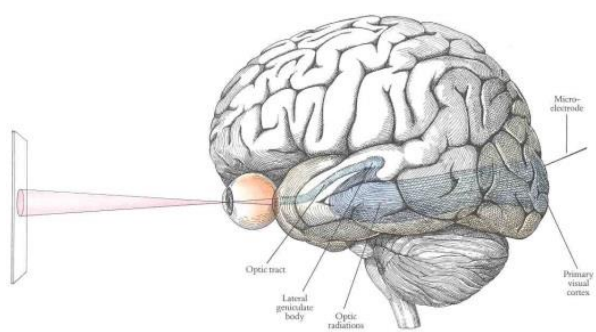
Coffee Time

Neuromorphic computing

Brain is not only intelligent but also energy efficient

Identify a cat

Google's 16,000 cores
~1,000,000 W



~20 W

Von Neumann architecture VS brain

Von Neumann architecture

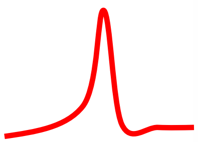
- Consists of
  - a processing unit
  - a control unit
  - a memory
  - external mass storage
  - input and output mechanisms
- The parts are mainly separate
- Mainly a serial system
- Good at processing precise logic sequences
  - Arithmetic computing
  - Word processing
  - etc.

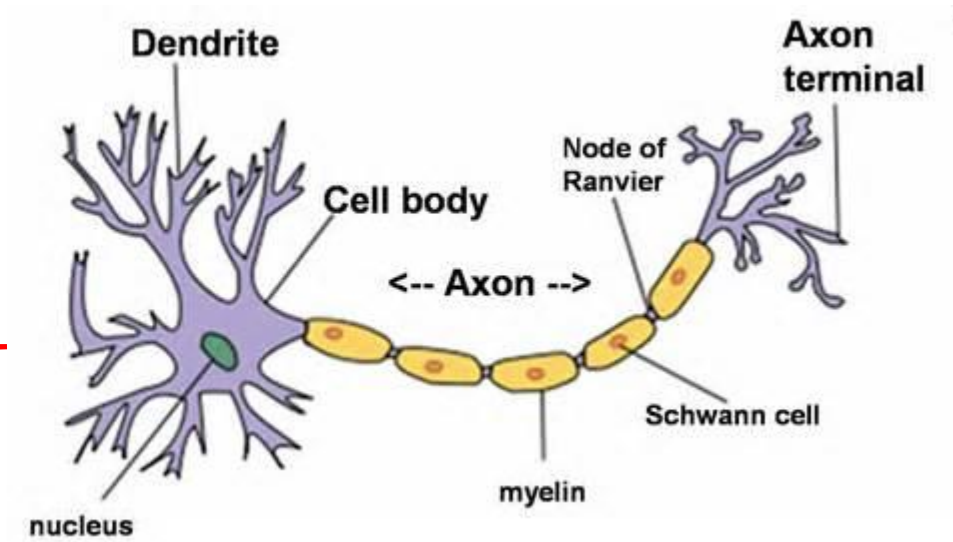
Brain

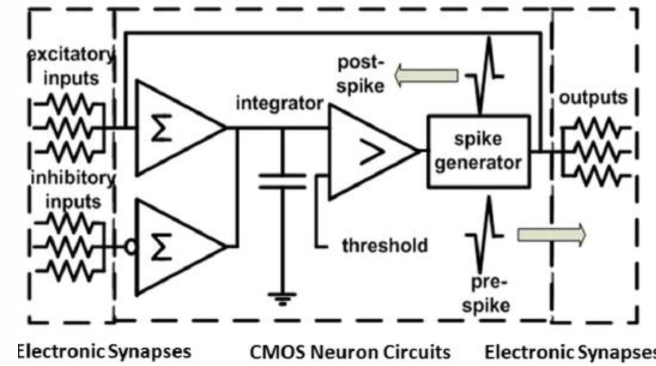
- Consists of
  - Sensory system including visual, auditory, somatosensory, etc.
  - Motor system
  - Memory system
  - Reasoning system
- Most parts are integrated together
- A massive parallel system
- Good at processing uncertain, dynamic and nonstructured info.
  - Visual perception
  - Language processing
  - etc.

Neuromorphic hardware-basic elements

spike







Yu, et al., 2011

Neuron

- Digital processors
  - DSP、GPU、FPGA
- Specific CMOS

Synapse

- Digital circuits
  - Large size and high energy
- Nanotech (memristors)
  - HP
  - IBM
  - Stanford Univ.

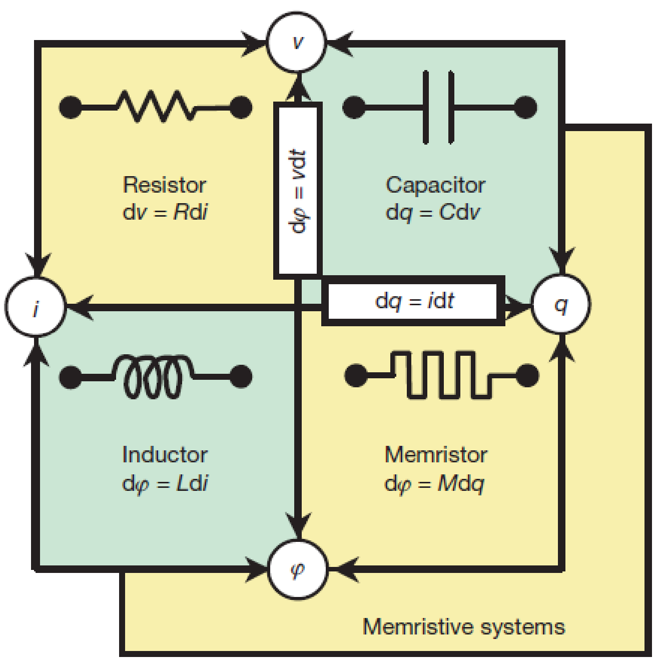
Memristor


nature
Vol 453|1 May 2008|doi:10.1038/nature06932

LETTERS

The missing memristor found

Dmitri B. Strukov[1], Gregory S. Snider[1], Duncan R. Stewart[1] & R. Stanley Williams[1]



- Predicted by Leon Chua, 1971
- Different types of memristors have been devised
- Can also simulate *Hodgkin-Huxley neurons* (neuristor, see Pickett et al., Nature Materials, 2013)

Large-scale neuromorphic computing systems

2008

- Supported by DARPA
- SRAM/memristor for simulating synapses
- CMOS for simulating neurons

2011

- The first generation of neurosynaptic core
- The core has 256 neurons and 256x1024 synapses
- Software: Compass

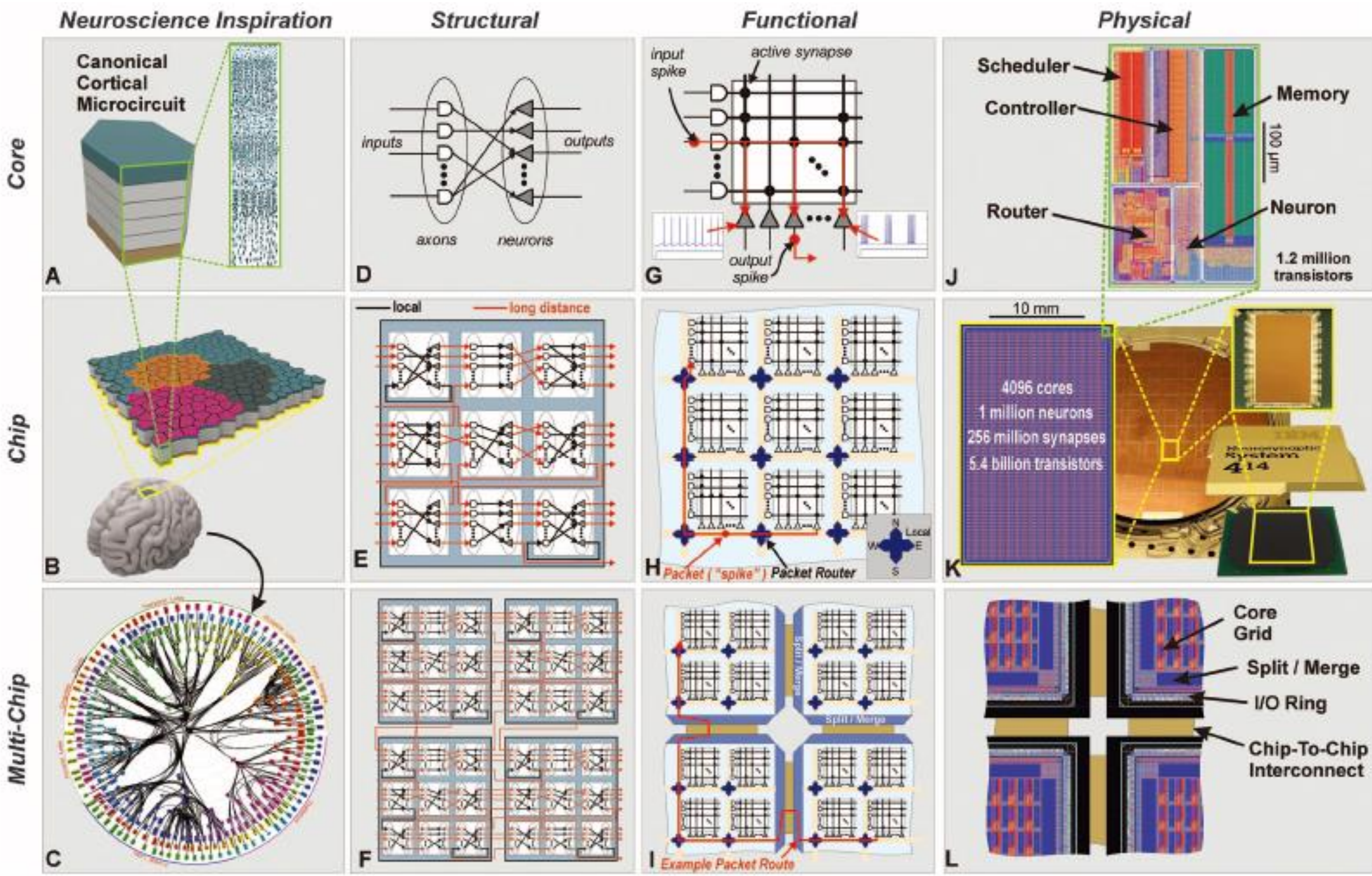
2014

- The second generation of neurosynaptic core
- Buit a chip "TrueNorth" with 4096 cores
- 1M programmable spiking neurons and 256M configurable synapses
- Multiobject detection and classification

SyNAPSE project

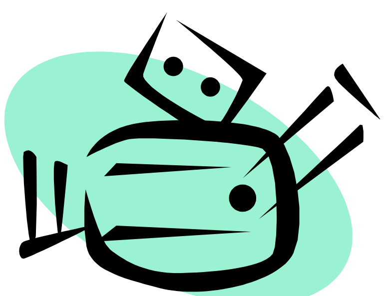
- Goal: build a non-von Neumann computing system
- Supported by DARPA

The TrueNorth architecture



Merolla, et al., Science, 2014

Demo



Watch a video

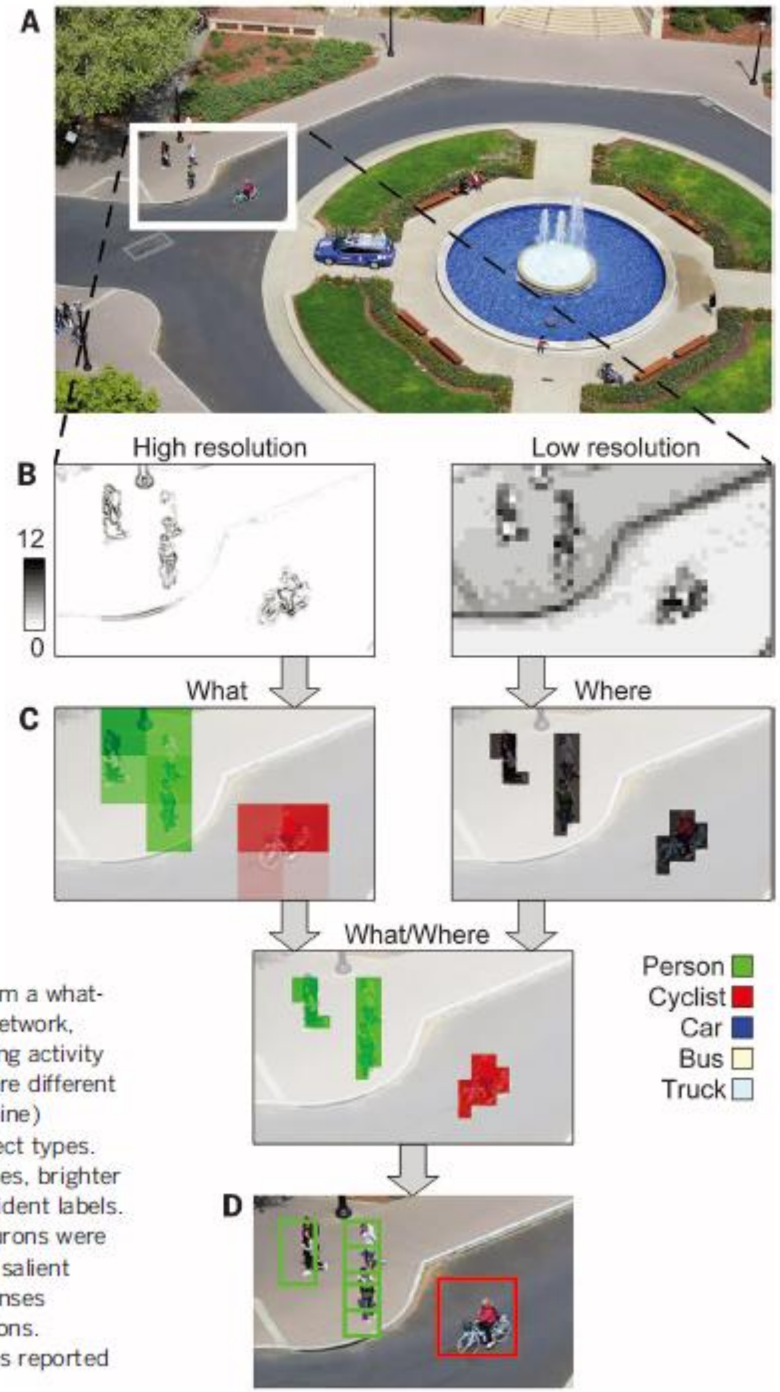
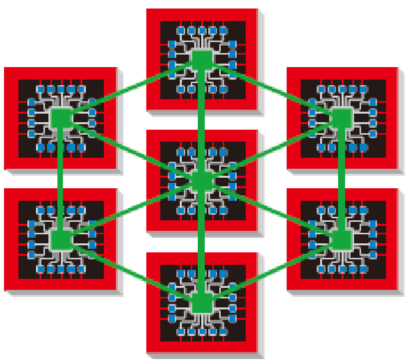
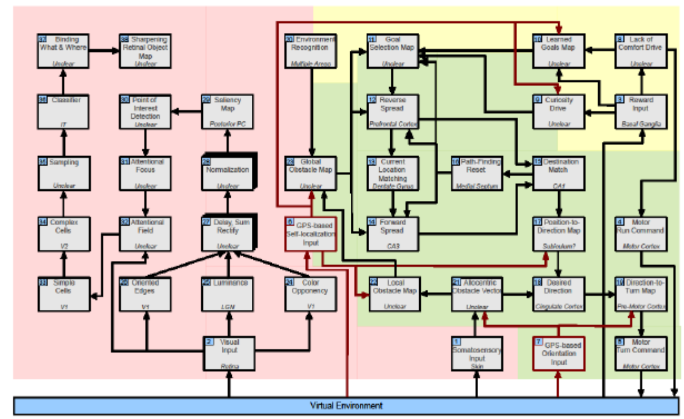


Fig. 3. Real-time multi-object recognition on TrueNorth. (**A**) The Neovision2 Tower data set is a video from a fixed camera, where the objective is to identify the labels and locations of objects among five classes. We show an example frame along with the selected region that is input to the chip. (**B**) The region is transduced from pixels into spike events to create two parallel channels: a high-resolution channel (left) that represents the what pathway for labeling objects and a low-resolution channel (right) that represents the where pathway for locating salient objects. These pathways are inspired by dorsal and ventral streams in visual cortex (*4*). (**C**) What and where pathways are combined to form a what-where map. In the what network, colors represent the spiking activity for a grid of neurons, where different neurons were trained (offline) to recognize different object types. By overlaying the responses, brighter colors indicate more-confident labels. In the where network, neurons were trained (offline) to detect salient regions, and darker responses indicate more-salient regions. (**D**) Object bounding boxes reported by the chip.

Other Systems





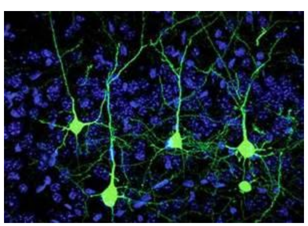
- 2008: use ARM core and SDRAM simulate neuron and synapse
- 2011: SpiNNaker chip with 18 ARM cores
- 2013: target 1036800 ARM cores
- Now: Implementing SPAUN and other systems

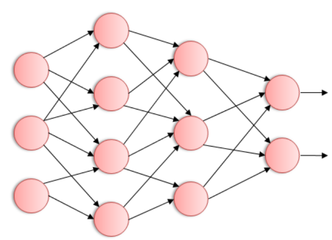
- 2008: invent memristor
- 2012: propose Cog ex Machina, a software framework for building massively-parallel applications on commodity, multicore hardware.

Neuromorphic computing in Tsinghua

2015.4.18

- School of Medicine
- Dept. of Precision Instrument
- Dept. of Computer Science and Technology
- Dept. of Electronic Engineering
- Dept. of Automation
- Dept. of Materials
- Institute of Microelectronics





Theory → Model → Hardware

Images from wiki

Neuromorphic computing in Tsinghua

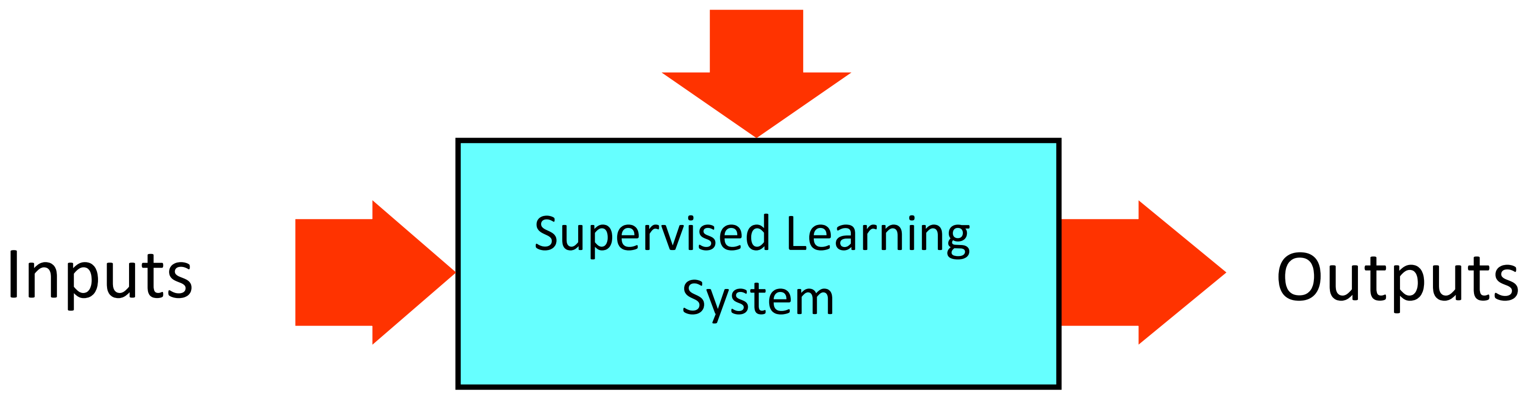
2016.4.15向李克强总理汇报

Outline

- Three types of learning
- Markov decision process
- Reinforcement learning

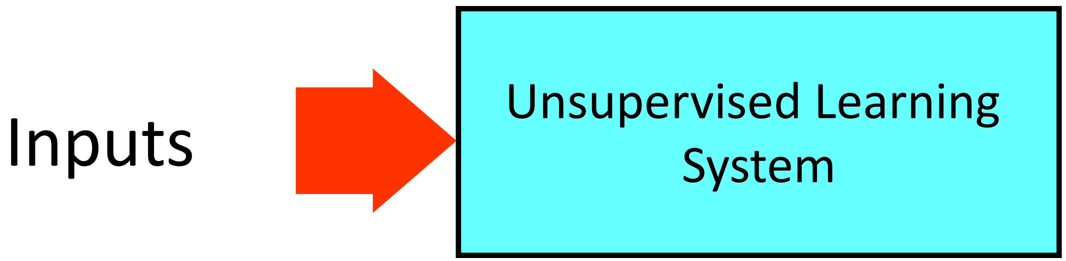
Supervised Learning

Training Info = desired (target) outputs



Error = (target output − actual output)

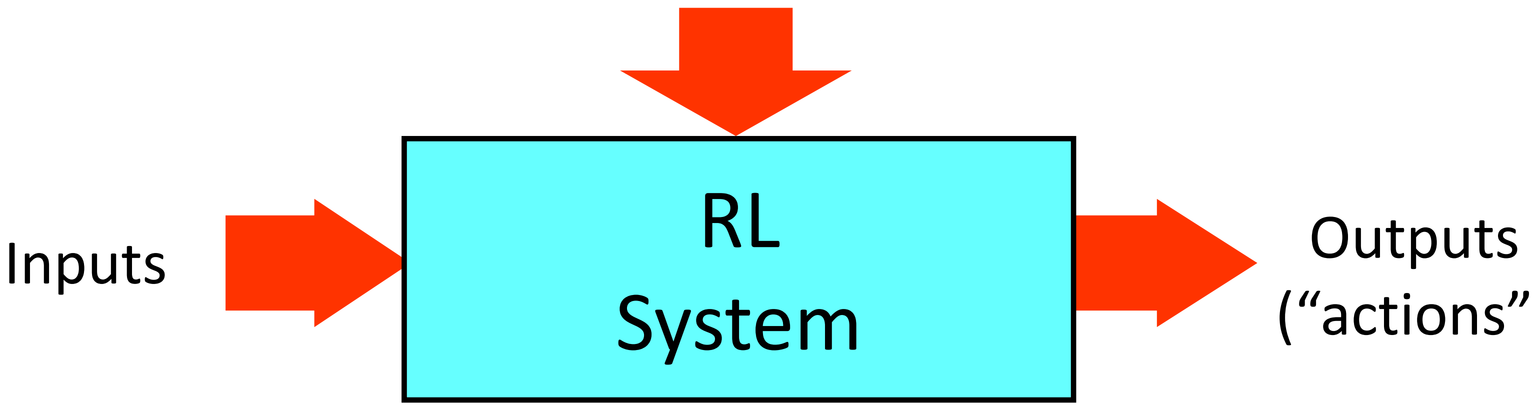
Unsupervised Learning



Objective: get another representation of input

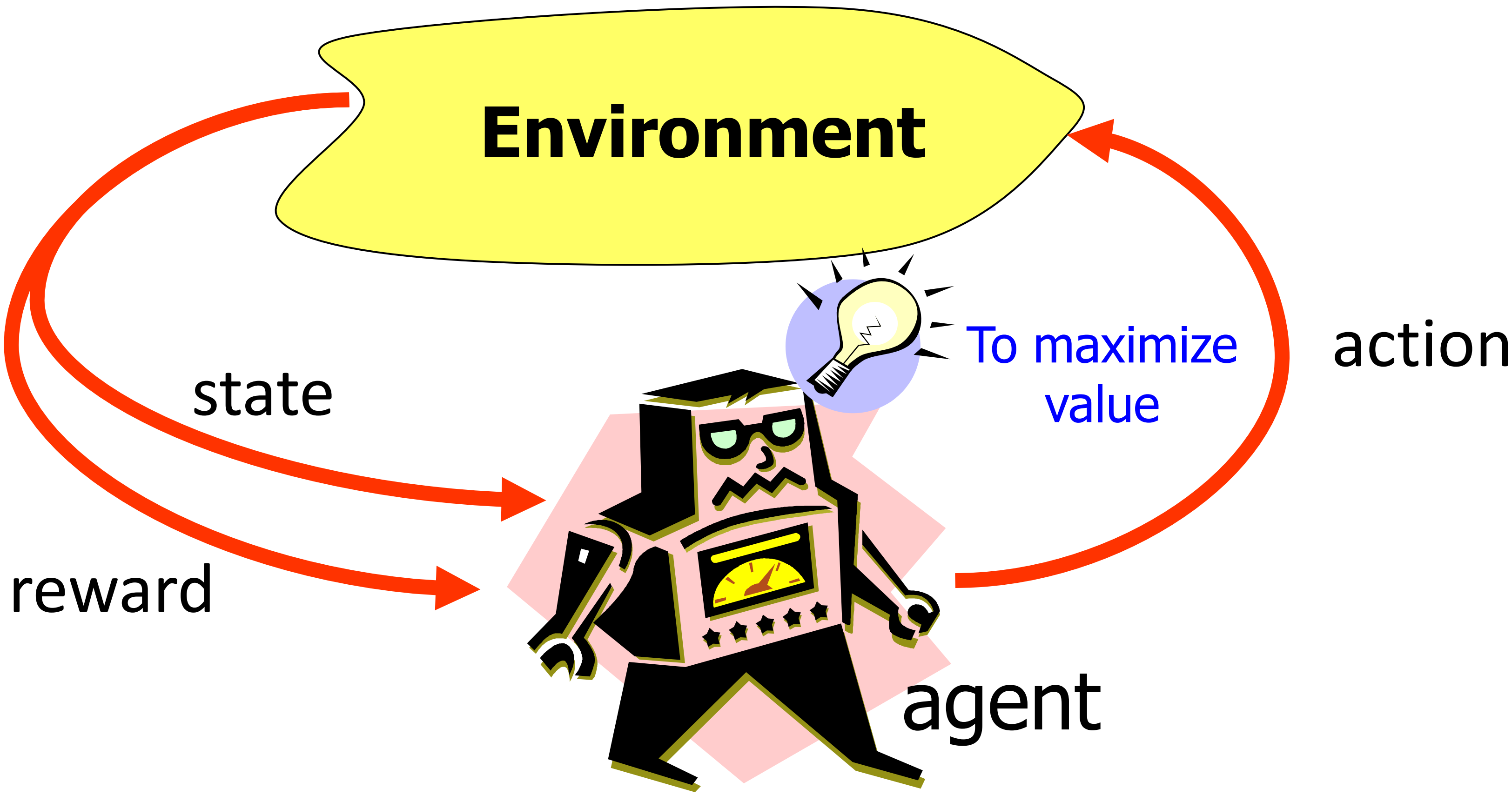
Reinforcement Learning

Training Info = evaluations (“rewards” / “penalties”)



Objective: get as much reward as possible

Main Elements of RL



Example (Bioreactor)

- State
  - current temperature and other sensory readings, composition, target chemical
- Actions
  - how much heating, stirring, what ingredients to add
- Reward
  - moment-by-moment production of desired chemical

Applications of reinforcement learning

- [Stanford autonomous helicopter]

Applications of reinforcement learning

26 Feb 2015

- Atari 2600 platform offers 49 games
- Google's deep Q-network (DQN) performs the same as or better than the human expert in 29 games

Applications of reinforcement learning

28 January 2016

- AlphaGo beat 欧洲围棋冠军樊麾(2015年10月5号)
- AlphaGo beat 世界围棋冠军李世石(2016年3月)

Outline

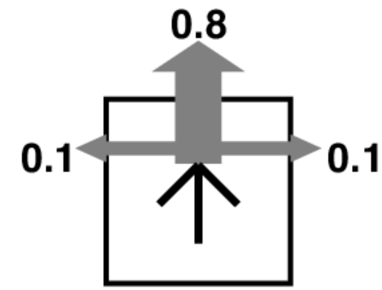
- Three types of learning
- Markov decision process
- Reinforcement learning

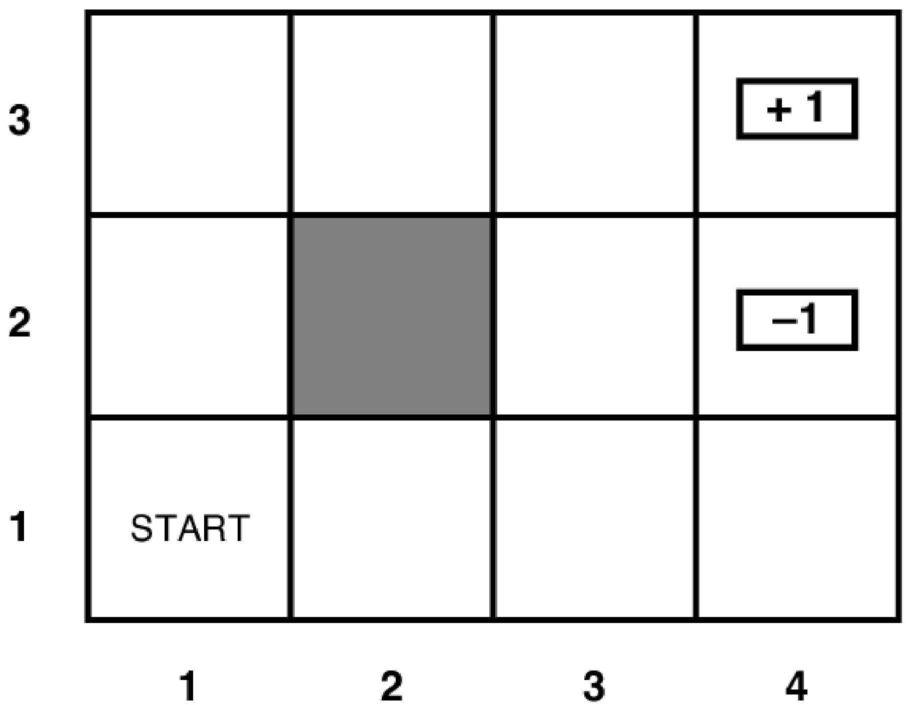
Markov Decision Process

- Components:
  - States s, beginning with initial state s_0
  - Actions a
    - Each state s has actions $A(s)$ available from it
  - Transition model $P(s'|s,a)$
    - Markov assumption: the probability of going to s' from s depends only on s and a and not on any other past actions or states
  - Reward function $\rho(s)$
- The "solution" to an MDP
  - Policy $\pi(s)$: the action that an agent takes in any given state

Example: Grid world

- Two terminal states. The gray patch denotes a wall.
- At every nonterminal state, there are four choices of actions {left, right, up, down}



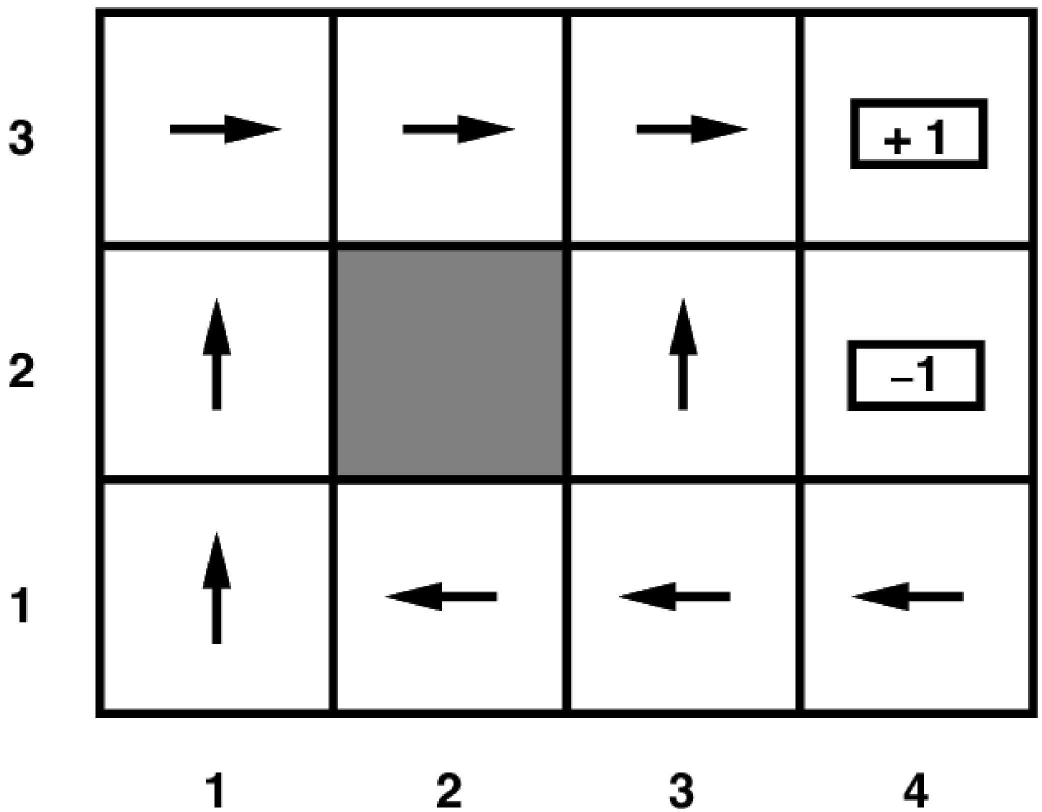


Transition model:

- The “intended” outcome occurs with probability 0.8, but with probability 0.2 the agent moves at perpendicular angles to the intended direction.
- A collision with a wall results in no movement.

Example: Grid world

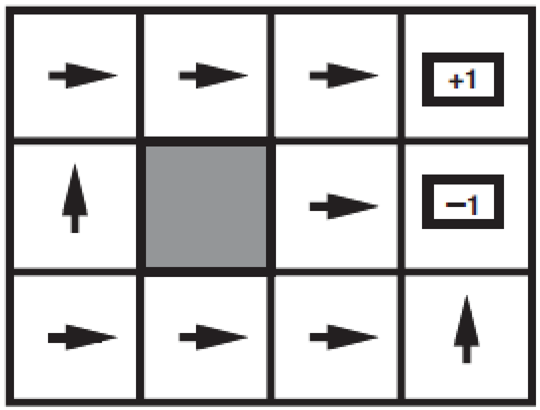
$R(s) = -0.04$ for every non-terminal state; ± 1 for the two terminal states



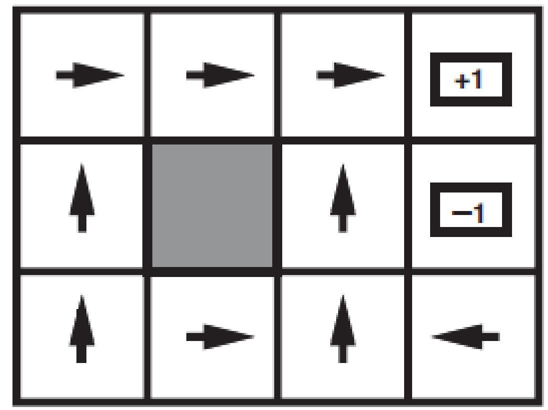
Optimal policy when $R(s) = -0.04$ for every non-terminal state

Example: Grid world

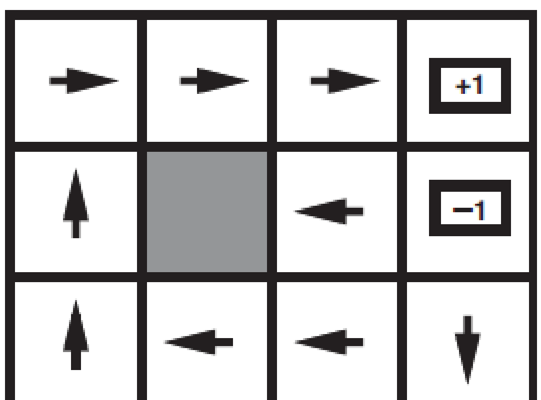
- Optimal policies for other values of $R(s)$:



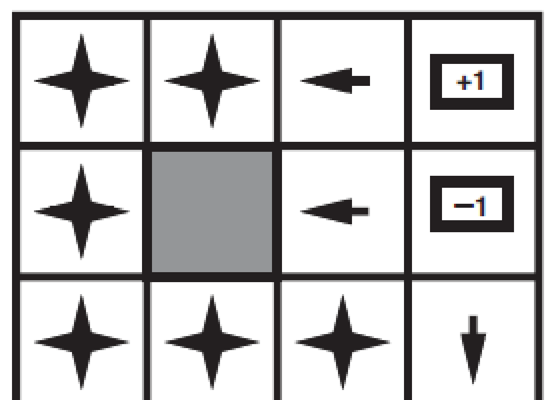
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$



$-0.0221 < R(s) < 0$



$R(s) > 0$

- The careful balancing of risk and reward is a characteristic of MDPs
- MDPs have been studied in several fields, including artificial intelligence, operations research, economics, and control theory

How to solve the MDP?

Maximizing expected reward

- For each possible policy π the agent might adopt, define the expected reward over states

$$\begin{aligned} V^{\pi}(s_0) &\equiv E(R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots) \\ &= E\left(\sum_{t=0}^{\infty} \gamma^n R(s_t)\right) \qquad\qquad 0 < \gamma \leq 1 \end{aligned}$$

where $R(s_0), R(s_1)$, … are samples of reward function $\rho(s)$ generated by following policy π starting at state s_0

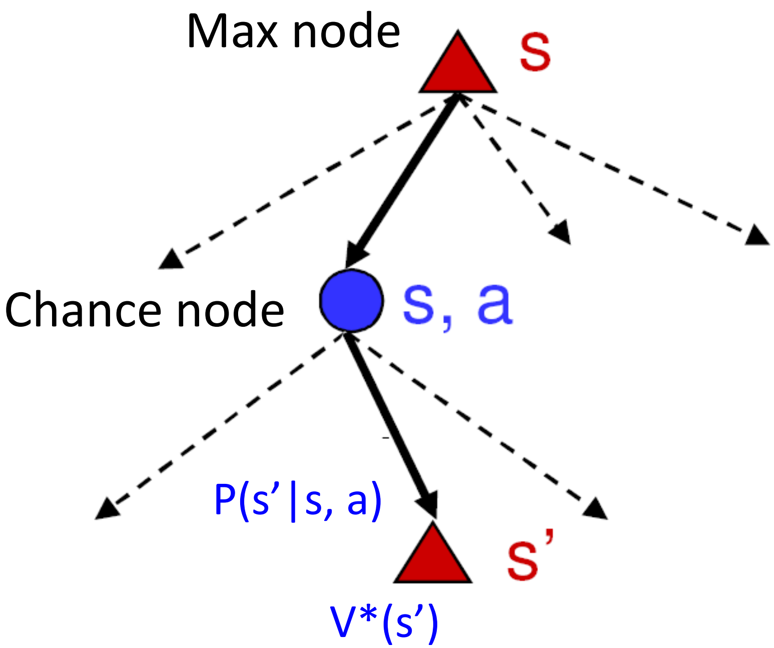
  – $R(s_0)$ is called "immediate reward"
  – In the Grid World example, $\gamma = 1$

- The optimal policy π^* maximize the expected reward

$$\pi^* \equiv \underset{\pi}{\operatorname{argmax}}\, V^{\pi}(s), (\forall s)$$

- Let $V^*(s)$ denote the value of V starting from s and following the policy π^*

Finding the rewards of states



- What is the expected reward of taking action a in state s?

$$\sum_{s'} P(s'|s,a)V^*(s')$$

- How do we choose the optimal action?

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a)V^*(s')$$

- What is the recursive expression for $V^*(s)$ in terms of the rewards of its successor states?

$$V^*(s) = E(R(s)) + \gamma \max_a \sum_{s'} P(s'|s,a)V^*(s')$$

where $E(\cdot)$ denotes expectation

The Bellman equation

- Recursive relationship between the rewards of successive states:

$$V^*(s) = E(R(s)) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) V^*(s')$$

- For N states, we get N equations for N unknowns
  - Solving them solves the MDP
  - Because this is a nonlinear system (due to the max operation), we solve them algebraically
  - Two methods: policy iteration and value iteration

Method 1: Value iteration

- Start out with every $V(s) = 0$
- Iterate until convergence
  - At each iteration, update the reward of each state according to this rule:

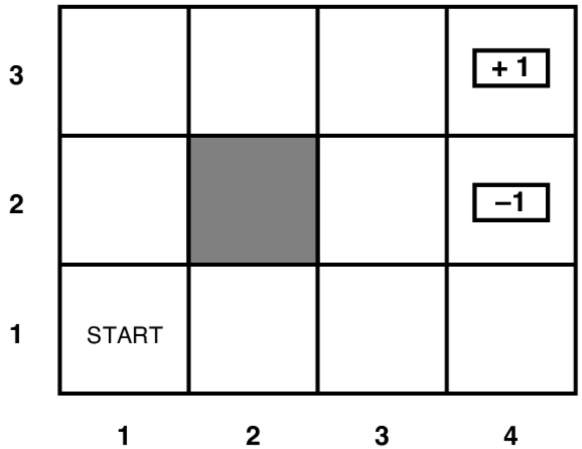
$$V(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) V(s')$$

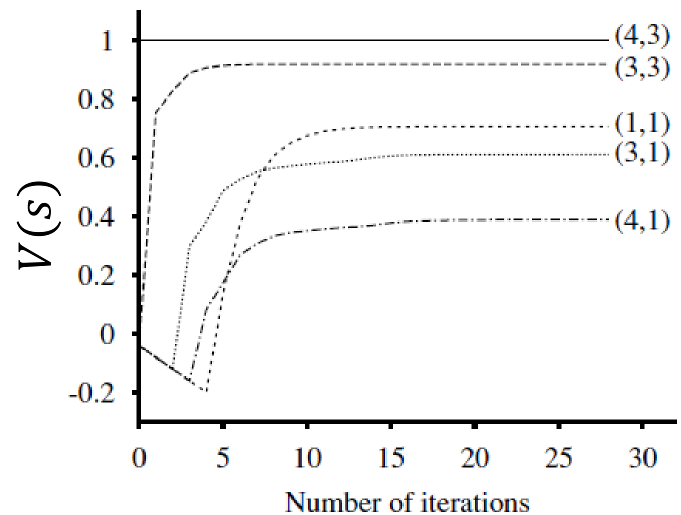
Inspired by the Bellman equation

- In the limit of infinitely many iterations, guaranteed to find the correct reward values
  - In practice, don't need an infinite number of iterations

Solving the Grid world example with value iteration

$R(s) = -0.04, \gamma = 1$

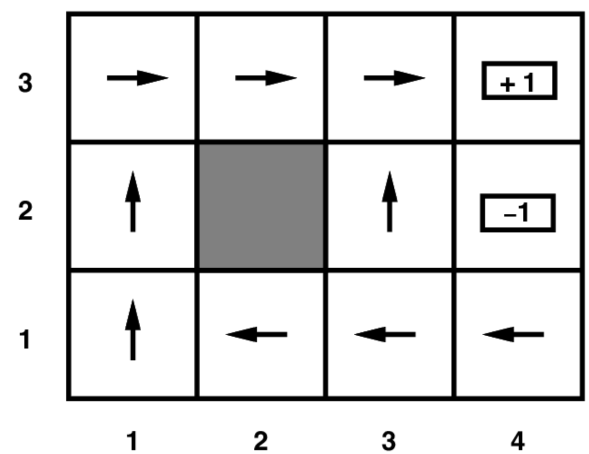




$V^*(s)$

3	0.812	0.868	0.918	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

$\pi^*(s)$



Method 2: Policy iteration

- Start with some initial policy π_0 and alternate between the following steps:
  - *Policy evaluation*: calculate $V^{\pi_i}(s)$ for every state
    $$V^{\pi_i}(s) = E(R(s)) + \gamma \sum_{s'} P(s' | s, \pi(s)) V^{\pi_i}(s')$$
    Can solve a linear system to get all the rewards!
  - *Policy improvement*: calculate a new policy π_{i+1}
    $$\pi_{i+1}(s) = \arg\max_{a \in A(s)} \left\{ \sum_{s'} P(s' | s, a) V^{\pi_i}(s') \right\}$$
- Because the number of policies is finite, this algorithm is bounded to converge

Outline

- Three types of learning
- Markov decision process
- Reinforcement learning

MDP vs RL

Regular MDP

- Given:
  - Transition model $P(s'|s, a)$
  - Reward function $R(s)$
- Find:
  - Policy $\pi(s)$

Reinforcement learning

- Transition model and reward function initially unknown
- Find
  - Policy $\pi(s)$
- "Learn by doing"

Imagine playing a new game whose rules you don't know; after a hundred or so moves, your opponent announces, "You lose." This is reinforcement learning.

Basic scheme

- In each time step:
  - Take some action
  - Observe the outcome of the action: successor state and reward
  - Update some internal representation of the environment and policy
  - If you reach a terminal state, just start over (each pass through the environment is called a *trial*)

Model-based learning versus model-free learning

- Model-based
  - Learn the model of the MDP (transition probabilities $P(s'|s,a)$ and rewards $\rho(s)$) and try to solve the MDP concurrently
- Model-free
  - Learn how to act without explicitly learning the transition probabilities $P(s'|s,a)$ and rewards $\rho(s)$
  - Q-learning ⟵ Value iteration in MDP
  - Actor-critic learning ⟵ Policy iteration in MDP

Q-learning

- Define an action-reward function $Q(s, a)$

$$Q(s,a) \equiv r(s) + \gamma \sum_{s'} P(s'|s,a)V^*(s')$$

where $r(s) = E\big(R(s)\big)$ and $V^*(s)$ is the expected reward with the optimal policy π^*

- Optimality

$$V^*(s) = \max_a Q(s,a) \qquad \pi^*(s) = \operatorname*{argmax}_a Q(s,a)$$

- Equilibrium constraint on Q values:

$$Q(s,a) = r(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

- If we know $P(s'|s, a)$, then we can solve Q using *value iteration*
- Problem: we don't know (and don't want to learn) $P(s'|s, a)$

Learn the Q-function

- For each s, a initialize table entry $\hat{Q}_0(s, a) \leftarrow 0$
- Observe current state s
- In iteration n, do:
  - Select an action a and execute it
  - Receive immediate reward r
  - Observe the new state s'
  - Update the table entry for $\hat{Q}_n(s, a)$
  - $s \leftarrow s'$

It follows the *value iteration* method in MDP

Temporal difference (TD) learning

- Training rule

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha_t(s,a)[\underbrace{R(s) + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)}_{\text{Temporal difference}}]$$

where

$$\alpha_t(s,a) = \frac{1}{1 + visits_t(s,a)}$$

Number of visits to (s,a) in t iterations

- Can prove convergence of $\hat{Q}$ to Q [Watkins and Dayan, 1992]

Intuition for this learning rule

We are trying to minimize $\left(Q - \hat{Q}\right)^2/2$. The ideal learning rule is

$$\hat{Q} \leftarrow \hat{Q} - \alpha_t \frac{\partial\left(Q - \hat{Q}\right)^2/2}{\partial \hat{Q}} = \hat{Q} + \alpha_t(Q - \hat{Q})$$

But Q is unknown, so we approximate it with $R(s) + \gamma \max_{a'} \hat{Q}(s',a')$

Learn the Q-function

- For each s, a initialize table entry $\hat{Q}_0(s, a) \leftarrow 0$
- Observe current state s
- In iteration n, do:
  - Select an action a and execute it
  - Receive immediate reward r
  - Observe the new state s'
  - Update the table entry for $\hat{Q}_n(s, a)$
  - $s \leftarrow s'$

Choose an action at state s

- **Option 1**: choose action a such that $\hat{Q}(s,a)$ is maximum

$\longrightarrow$ Early stopping

- **Option 2**: choose a stochastically such that the actions with higher $\hat{Q}(s,a)$ has higher probability to be chosen

$$\Pr(a|s) = \frac{k^{\hat{Q}(s,a)}}{\sum_b k^{\hat{Q}(s,b)}} \quad \text{where } k > 0$$

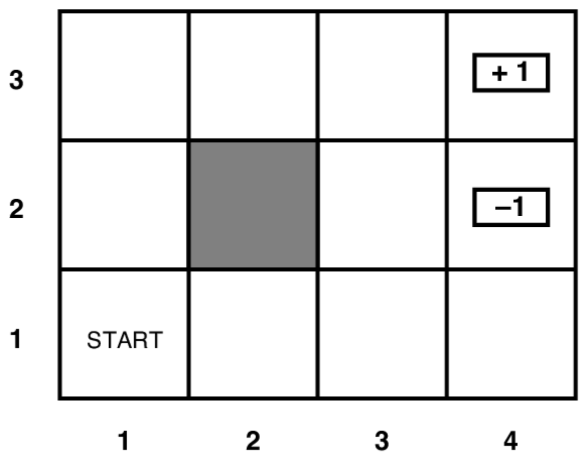
Exploitation

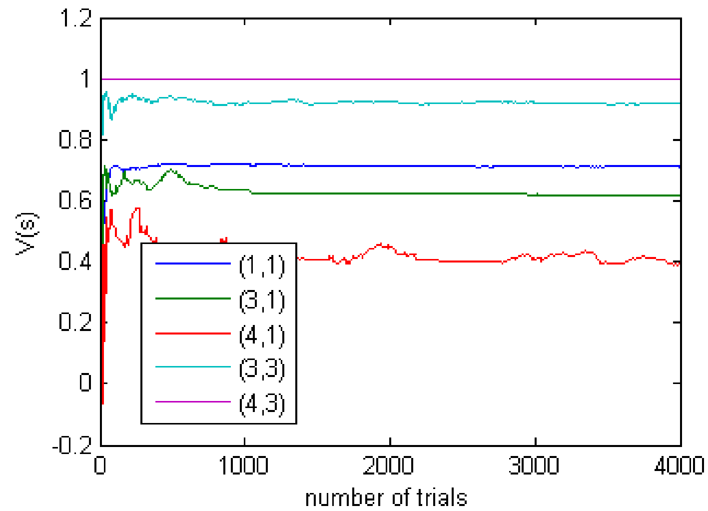
  - Larger k tends to choose actions with larger $\hat{Q}$ values
  - Smaller k allows to choose actions with smaller $\hat{Q}$ values
  - k can change with time

Exploration

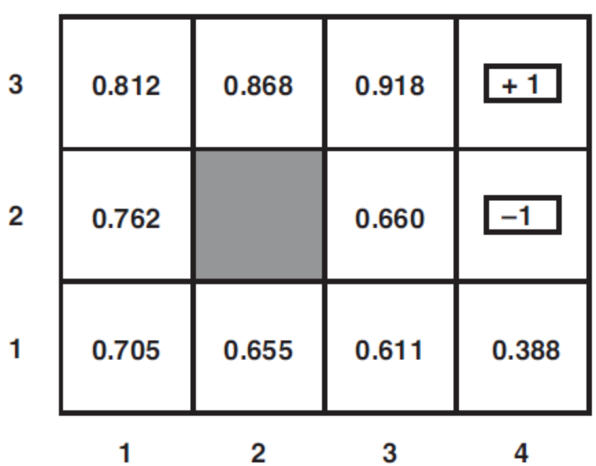
Solving the Grid world example with Q-learning

$R(s) = -0.04, \gamma = 1$

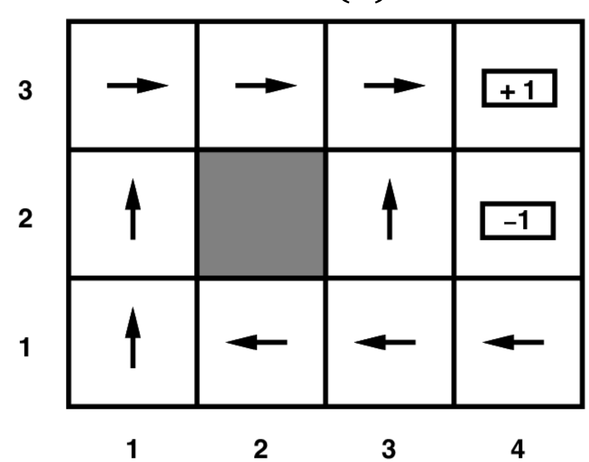




$V^*(s)$



$\pi^*(s)$



Actor-critic learning

Actor: a function mapping states to actions $\pi: S \rightarrow A$

Critic: a value function $V^{\pi}(s)$

- The actor executes a policy and the critic evaluates this policy
- It follows the *policy iteration* method in MDP
- Problem: we don’t know (and don’t want to learn) $P(s'|s,a)$
- A temporal difference (TD) method

Critic

Evaluate a fixed policy $\pi(s)$ (deterministic or stochastic)

- When a transition occurs from s to s', update the value function

$$V(s) \leftarrow V(s) + \alpha_t \delta$$

where $\epsilon > 0$ is the learning rate and δ is the *TD error*

$$\delta = R(s) + \gamma V(s') - V(s)$$

- Conditions are known under which $V(s)$ converges to $V^{\pi}(s)$

Same intuition applies to Q-learning

Intuition for this learning rule

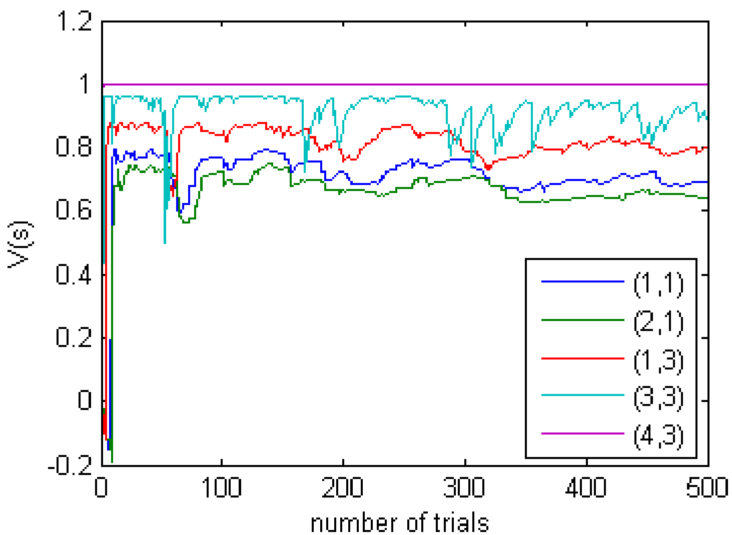
We are trying to minimize $(V^{\pi}(s) - V(s))^2/2$. The ideal learning rule is

$$V(s) \leftarrow V(s) - \alpha_t \frac{\partial (V^{\pi} - V)^2/2}{\partial V} = V(s) + \alpha_t (V^{\pi}(s) - V(s))$$

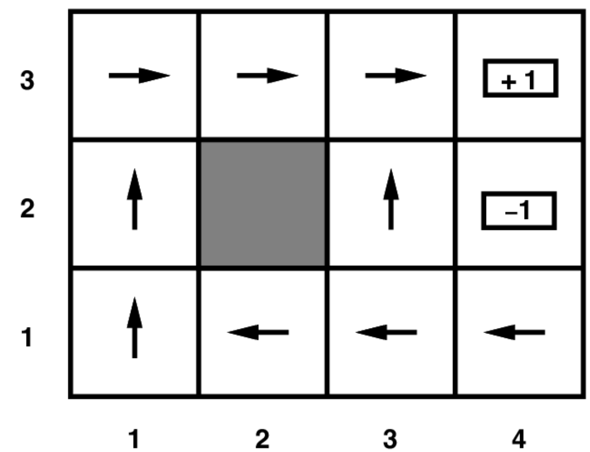
But $V^{\pi}(s)$ is unknown, so we approximate it with $R(s) + \gamma V(s')$

The Grid world example

- Let $R(s) = -0.04, \gamma = 1$
- Evaluate the policy on the right (this policy happens to be the optimal)
- The results with $\alpha_t = 60/(59 + visits_t(s))$ are shown below, where $visits_t(s)$ denotes the number of times the agent has visited s



Deterministic $\pi^*(s)$



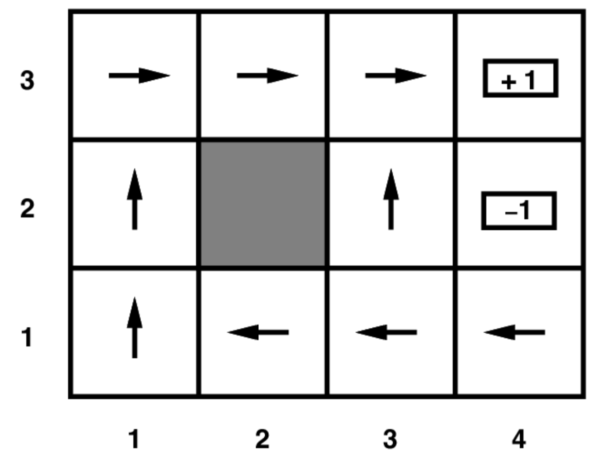
$V^*(s)$

	1	2	3	4
3	0.812	0.868	0.918	+1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388

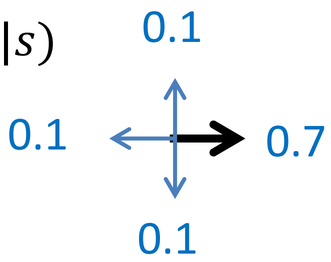
The Grid world example

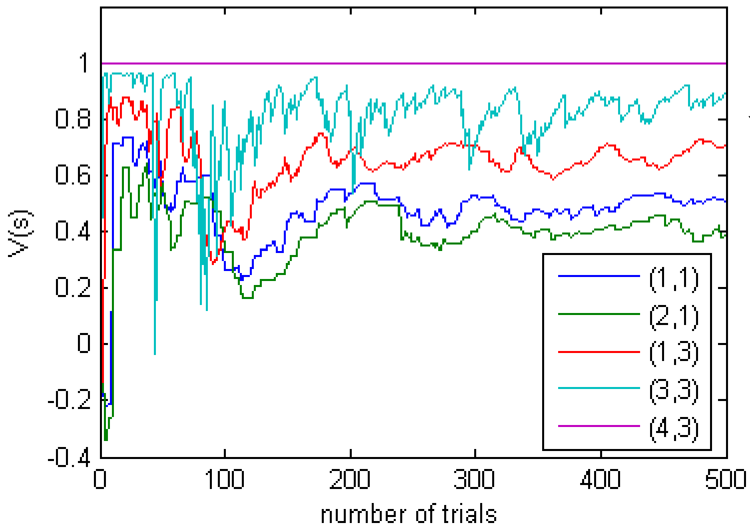
- Now evaluate a stochastic policy as follows
  - Select the action at s indicated below with probability 0.7
  - Select other three actions with probability 0.1

Stochastic $\pi_a(s)$



$P(a|s)$





The estimates get lower than the deterministic case

Action-reward function

- Define an action-reward function $Q^\pi(s,a)$ for policy π
$$Q^\pi(s,a) \equiv r(s) + \gamma \sum_{s'} P(s'|s,a) V^\pi(s')$$
where $V^\pi(s)$ is the expected reward with the policy π.
- Clearly $V^\pi(s)$ is the average value of the actions specified by policy π

$$V^\pi(s) = \sum_a \pi_a(s) Q^\pi(s,a) \text{ where } \pi_a(s) = \Pr[a|s]$$

Note the difference between $Q^\pi(s,a)$ and $Q(s,a)$ defined in Q-learning

$$Q(s,a) \equiv r(s) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$$

$$V^*(s) = \max_a Q(s,a)$$

Actor

Improve the policy $\pi(s)$

- Actions are generated by softmax method

$$\pi_a(s) = \Pr[a|s] = \frac{\exp(\beta m_a)}{\sum_b \exp(\beta m_b)}$$

- After $V(s)$ converges to $V^{\pi}(s)$ with the "critic" algorithm

$$V(s) = V^{\pi}(s) = \sum_a \pi_a(s) Q^{\pi}(s,a)$$

- Define the error

$$\tilde{\delta} = Q^{\pi}(s,a) - V(s)$$

- If the error is positive, selection of a should be strengthened; otherwise weakened.
- It can be shown that the following rule implements an approximate stochastic gradient ascend method (Dayan and Abbott, 2001)

$$\begin{cases} m_a & \leftarrow m_a + \alpha(1 - \pi_a(s))\tilde{\delta} \\ m_{a'} & \leftarrow m_{a'} - \alpha \pi_{a'}(s)\tilde{\delta}, \quad \forall a' \neq a \end{cases}$$

when a is selected, where $\alpha > 0$ is the learning rate.

Approximate the action-reward function

$$Q^\pi(s,a) \equiv r(s) + \gamma \sum_{s'} P(s'|s,a) V^\pi(s')$$

- The action-reward function is unknown as the transition model is unknown
- Let's approximate it with a sample $R(s) + \gamma V^\pi(s')$ resulted from executing action a at state s (according to the policy $\pi(s)$), and in the sequel approximate $\tilde{\delta}$ with

$$\delta = R(s) + \gamma V(s') - V(s)$$

The TD error used by the "critic" also

- Then the updating rule becomes

$$\begin{cases} m_a & \leftarrow m_a + \alpha(1-\pi_a(s))\delta \\ m_{a'} & \leftarrow m_{a'} - \alpha\pi_{a'}(s)\delta, \quad \forall a' \neq a \end{cases}$$

Optimality

- Learning is based on following and simultaneously trying to improve a policy.
- Normally, the action values are changed before the critic has converged
- There is no proof that the combined estimation and optimization procedure is guaranteed to find an optimal policy
- Nevertheless, it is found to work well in practice.

Summary of key points

- Three types of learning
- Markov decision process
  - Value iteration
  - Policy iteration
- Reinforcement learning
  - Model-based vs. model-free
  - Q-learning
  - Actor-critic learning

Further reading

- Russell, Norvig, 2010

  Chapters 21 of “Artificial Intelligence A Modern Approach (3rd Edition)”
- Dayan, Abbott, 2001

  Chapter 9 of “Theoretical Neuroscience”
- Dayan, Watkins, 2001

  Reinforcement Learning

  Encyclopedia of Cognitive Science