
welcome
to Lecture 10
Computation and the Brain

What happened last Wednesday

The Big Picture: Computation in the brain: What is the right level?

- Whole brain?

?

- Spiking neurons and synapses?
- Dendrites?
- Molecules?

"…we do not have a logic for the transformation of neural activity into thought and action. I view discerning [this] logic as the most important future direction of neuroscience…"

R. Axel *Neuron*, Sep 2018

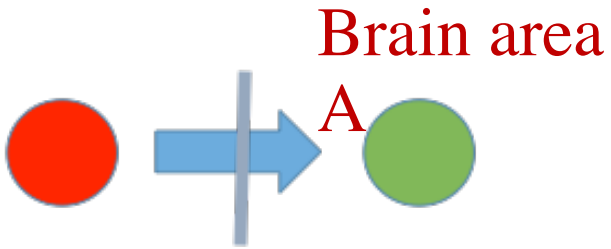
The assembly hypothesis

- There is an intermediate level of brain computation
- Involved in carrying out higher cognitive functions such as reasoning, planning, language, story-telling, math, music…
- Assemblies are its main "data structure"
- Fundamental operations: project, associate, merge
- Plus control operations: loops, conditionals, read, fire, disinhibit, enable…

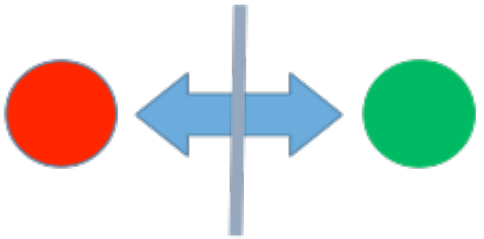
The model

- A finite number of brain areas **A**, **B**, …
- Each with n (excitatory) neurons ≈ 106-8
- E – I balance: assemblies have k neurons each ≈103-4
- NB: $k \approx \sqrt{n}$ (so assemblies barely intersect)
- All areas have recurrent connections
- Some areas have synaptic connectivity to others
- Have to be enabled (disinhibited)

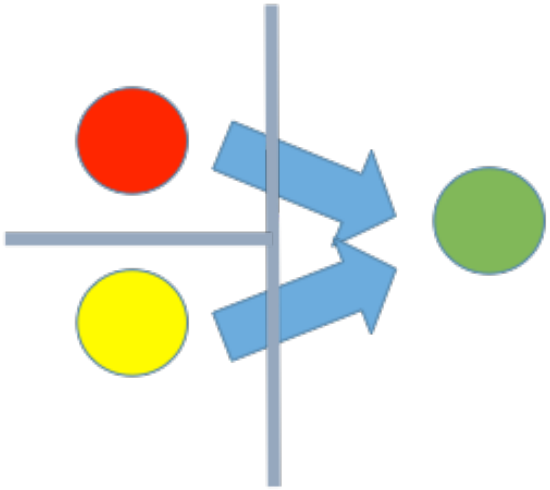
project(x, A, y)



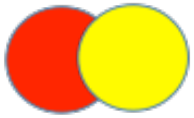
- Also:
- reciprocal_project(x, A, y)



- merge(x, y, A, z)



- associate(x, y) (same area)



The assembly hypothesis: what we know

- The basic operations are plausible (shown by theorems and simulations that they can be implemented)
- Similarity of assemblies is preserved under projection
- Modes of computation by assemblies
  - Algorithmic: Turing complete
  - Pattern completion: associative, probabilistic computation
  - Learning: so far, half spaces
- Language: syntax trees in the Brain

The assembly hypothesis: challenges

- Other operations? (NB: plausible and useful)
- Improve and extend simulations
- Predictive learning?
- Experimental validation/falsification?

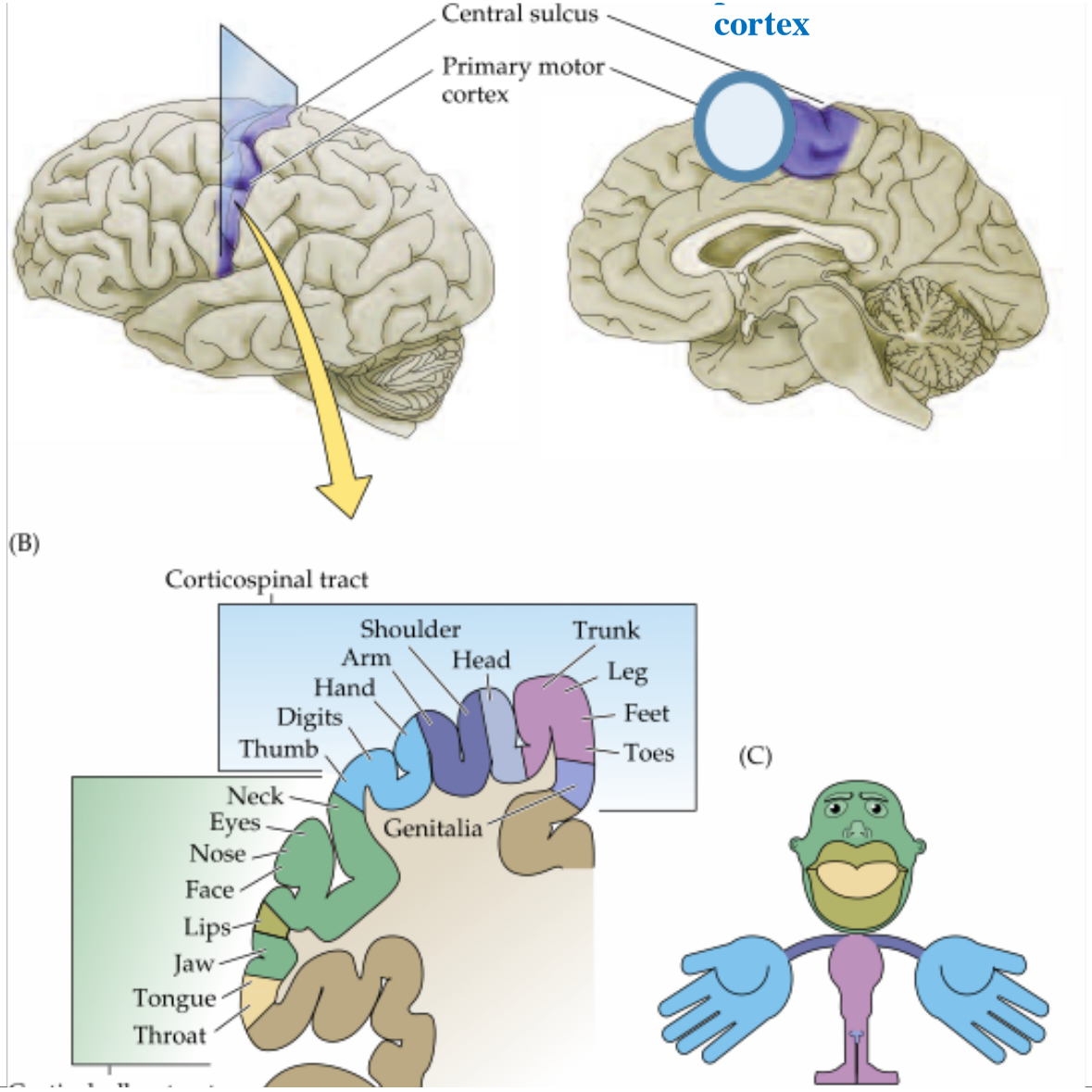
collaborators

Santosh Vempa
Georgia Te

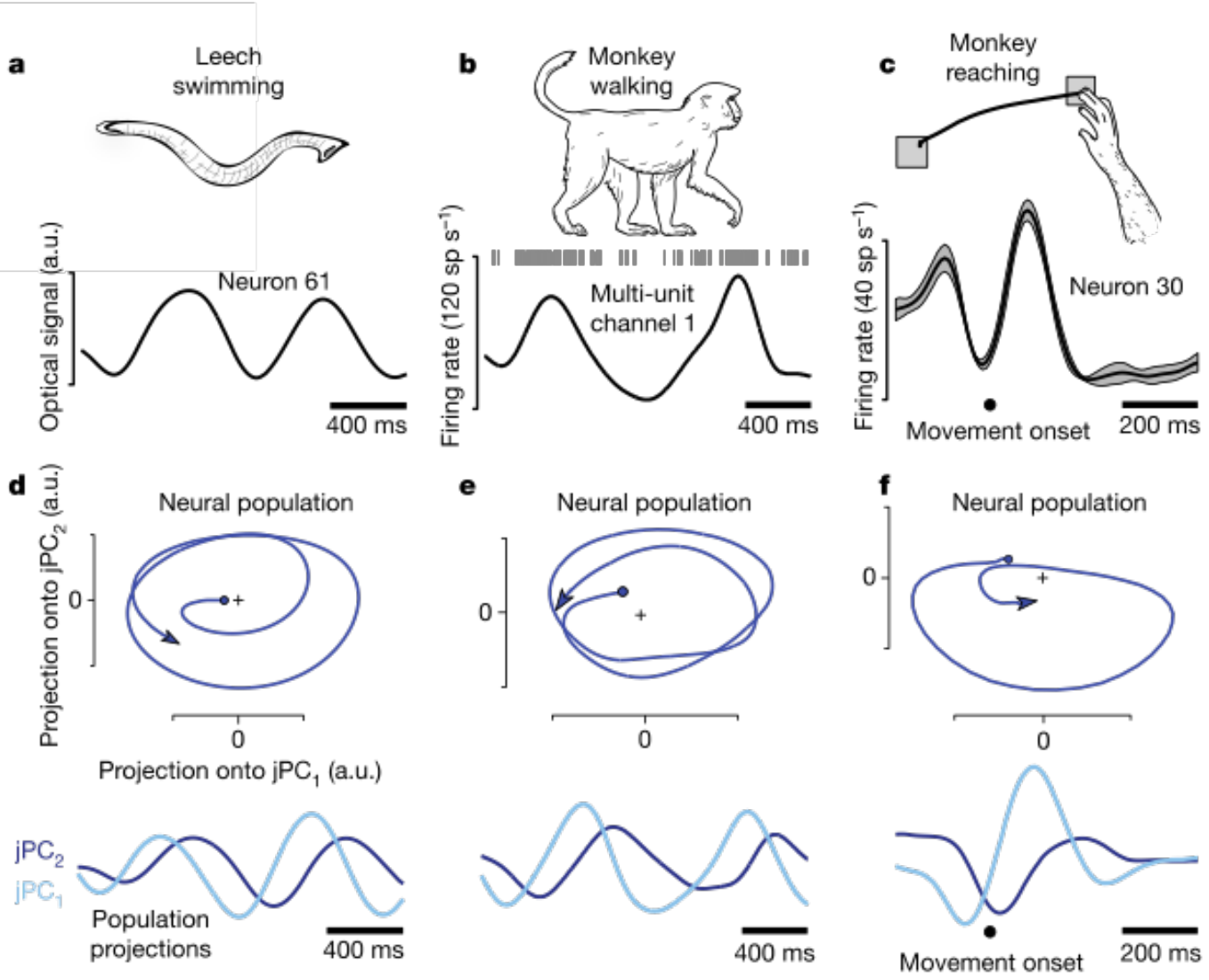
Wolfgang Maass

Mike Collins
Columbia U

The Motor Cortex

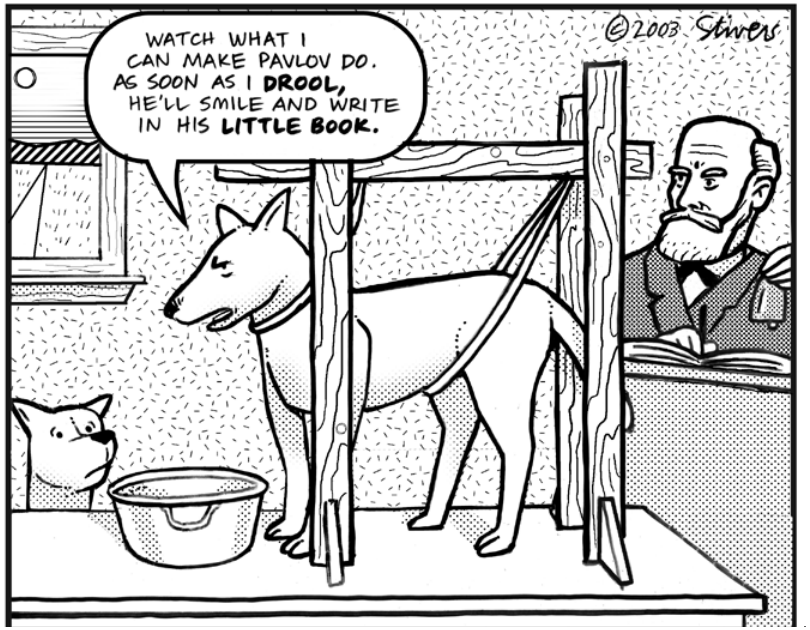


- Two animals
- Three motions
- Two of the motions are rhythmic, the third is not
- All have significant cyclic projections of neural activity
- ***jPCA***



Today

- Reinforcement learning
- Conditioning vs Temporal Differences
- RL in the Brain
- The math of RL
- Deep RL and Alpha Go

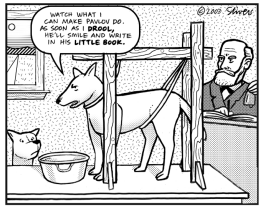


Next weeks

- Language
- Evolution and Development
- Discussion of Consciousness
- (Preliminary, volunteer) project presentations

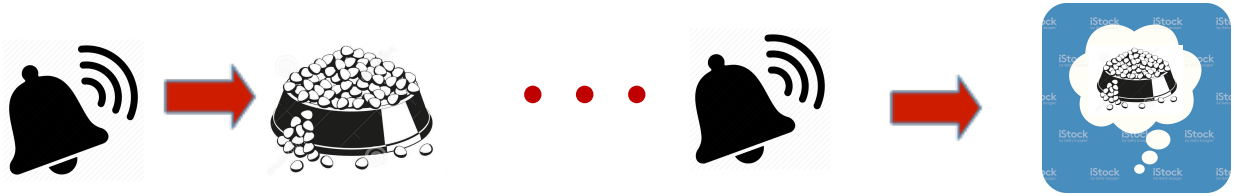
Reinforcement learning in the Brain

- Learning about the world, or about your actions, exclusively through rewards
- Rewards: positive or negative: food vs. work
- Conditioning, pre-1980s: how do past rewards affect animal expectations?
- Reinforcement learning: how do rewards affect animal behavior?



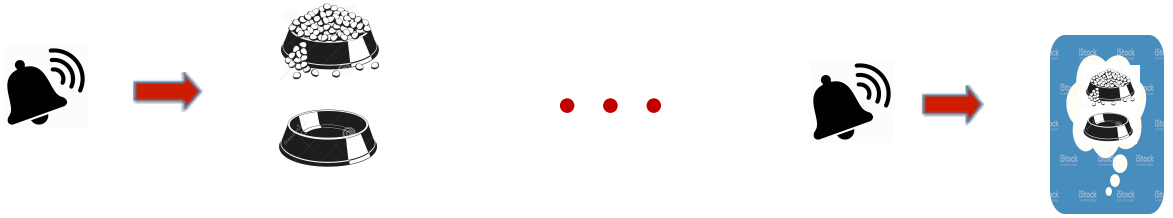
Classical conditioning

- Pavlovian

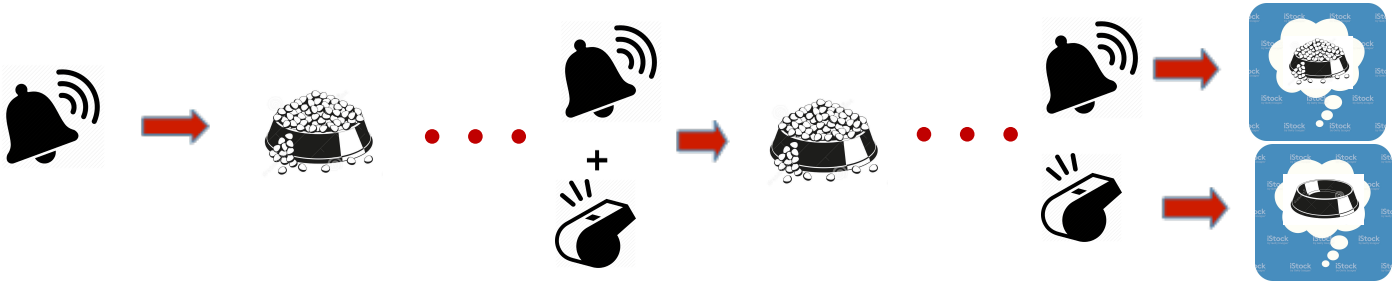


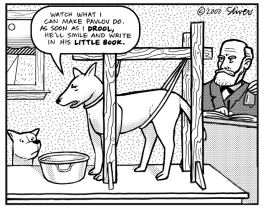
- Extinction

- Partial



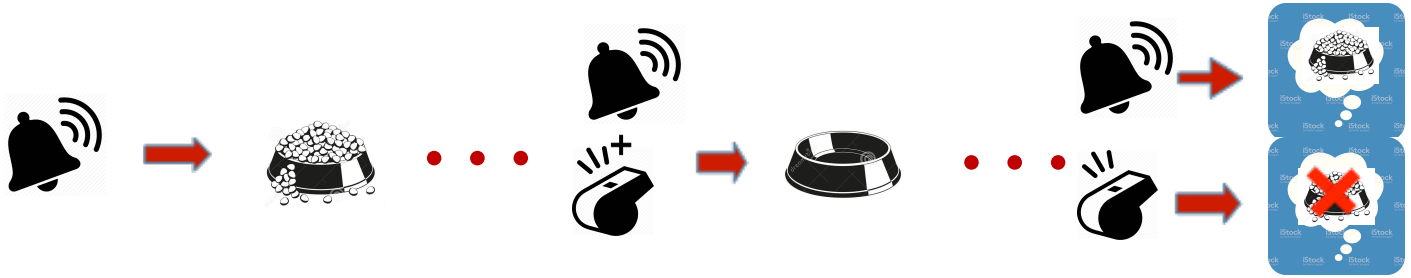
- Blocking



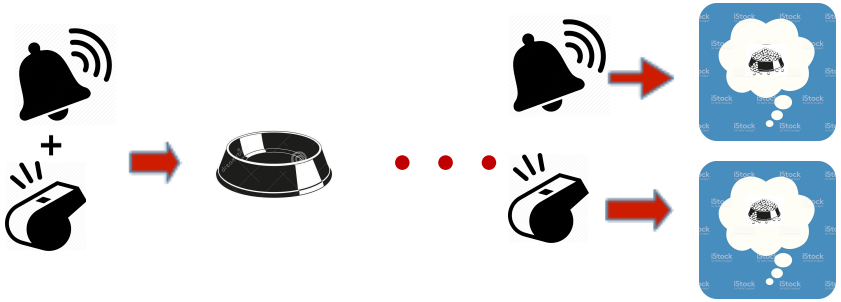


Classical conditioning (cont.)

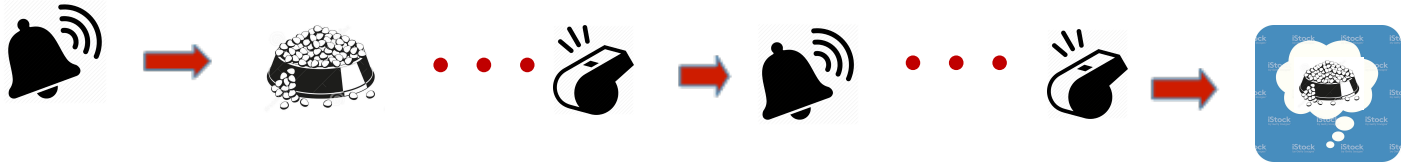
- Inhibition



- Overshadow



- Aristotelian



Explanation (1960s):
Rescola-Wagner update (delta rule)

Stimulus u, prediction x, weights w

(w, u possibly vectors)

Reward R

$x = u \cdot w$

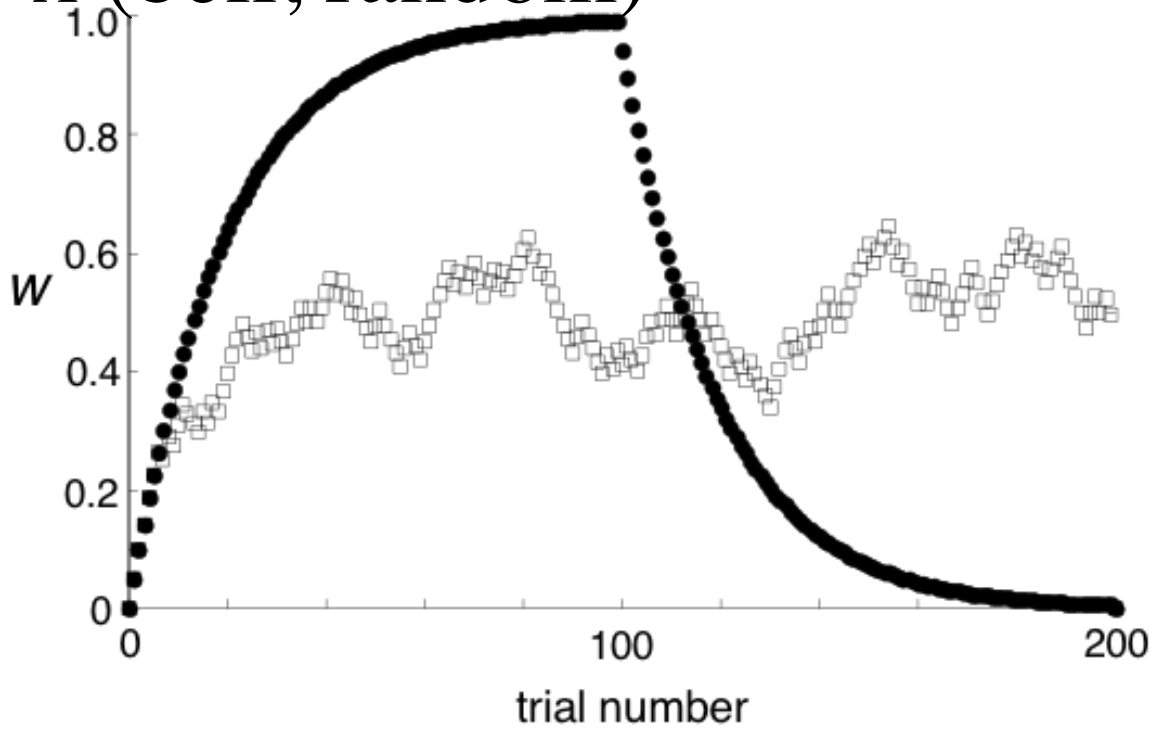
Rescola-Wagner plasticity: $\mathbf{w \rightarrow w + \varepsilon \cdot (R - x) \cdot w}$

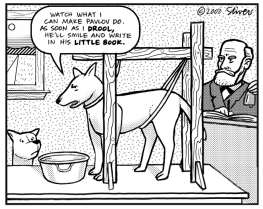
δ

Note that this is gradient descent with error $2(R - x)2$

It explains classical conditioning

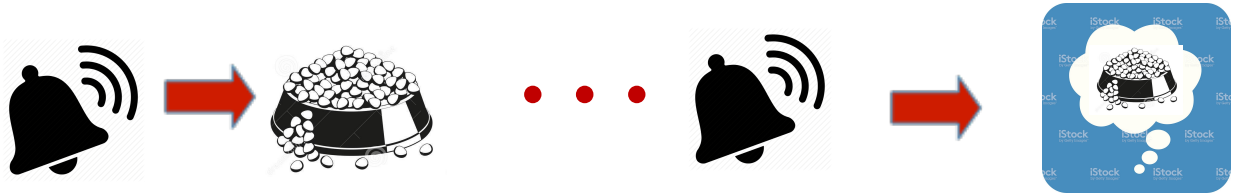
100 x (bell, food) + 100 x (bell, no food)
vs 200 x (bell, random)





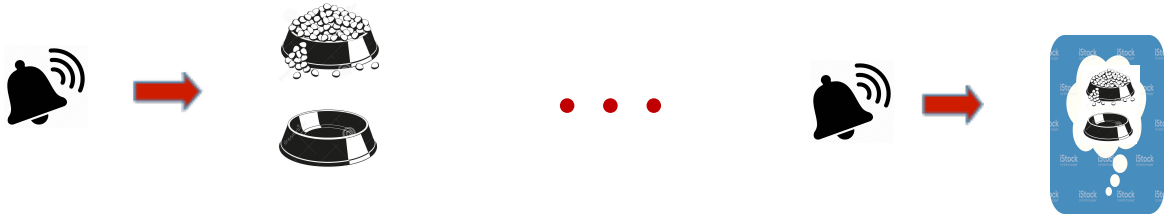
Classical conditioning

- Pavlovian

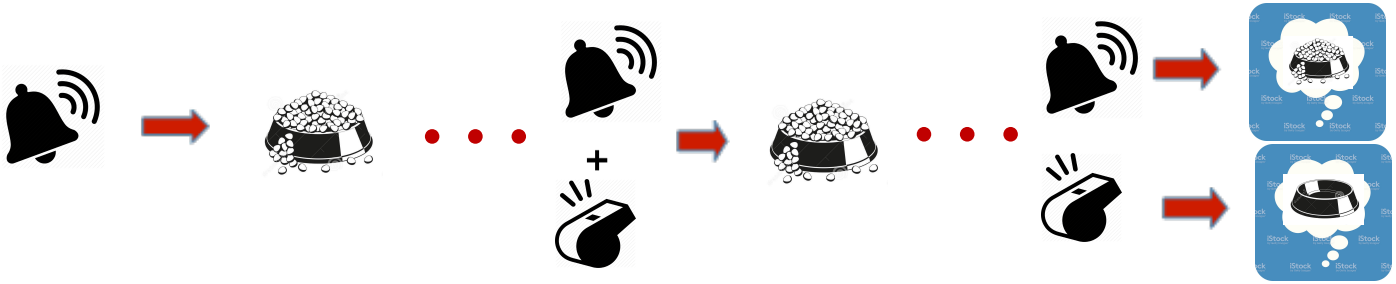


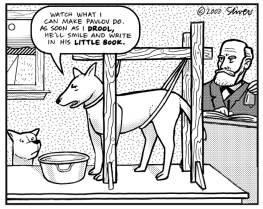
- Extinction

- Partial



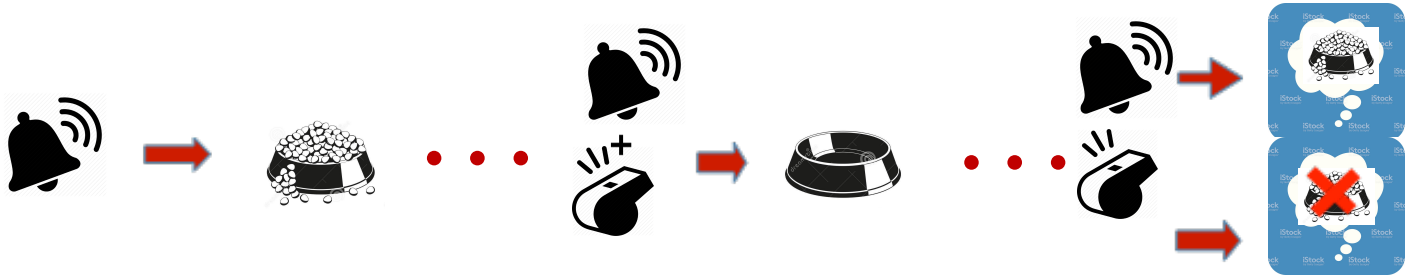
- Blocking



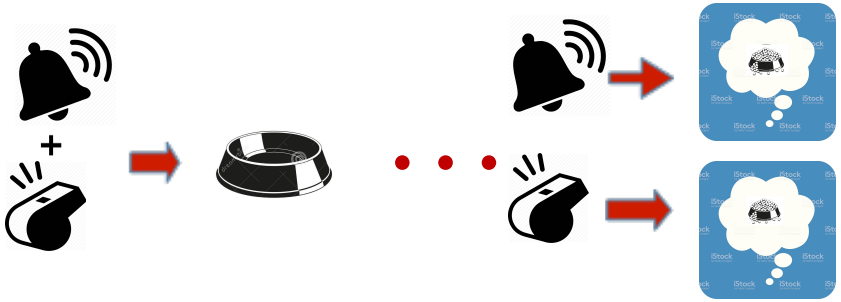


Classical conditioning (cont.)

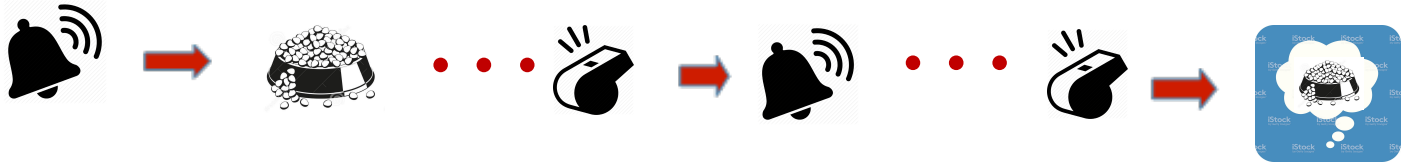
- Inhibition

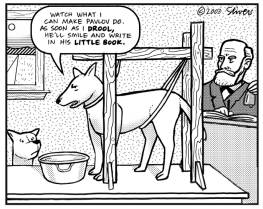


- Overshadow



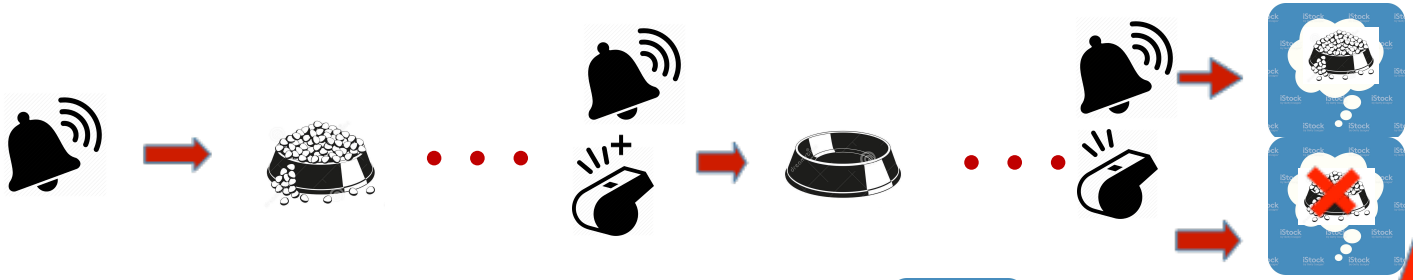
- Aristotelian



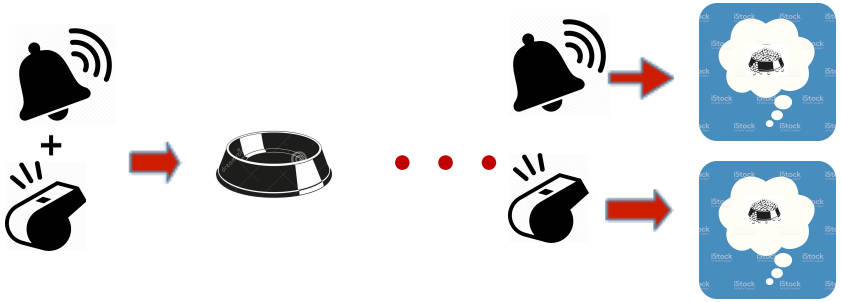


Classical conditioning (cont.)

- Inhibition

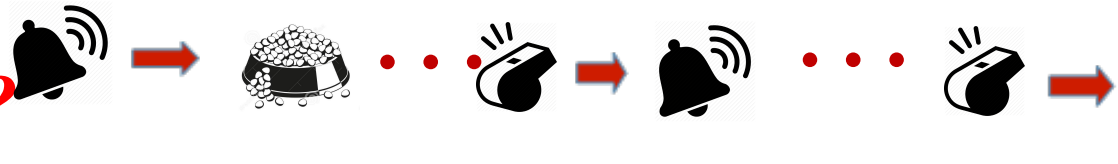


- Overshadow



timing !

Aristotelian?



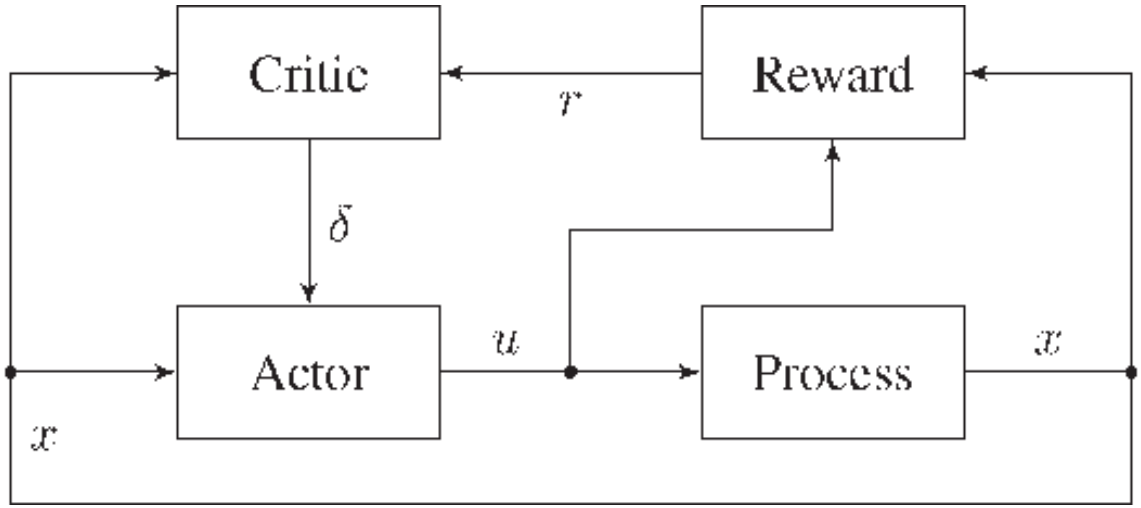
Even worse: no foresight

Animals choose actions
by looking beyond
the present reward…

It works! [Tesauro 1992] backgammon

before there was Alpha Go, there was TD-Gammon

Reinforcment learning:
The actor – critic formulation



Reinforcement Learning in the brain

Spot:

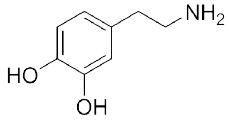
The δ calculator

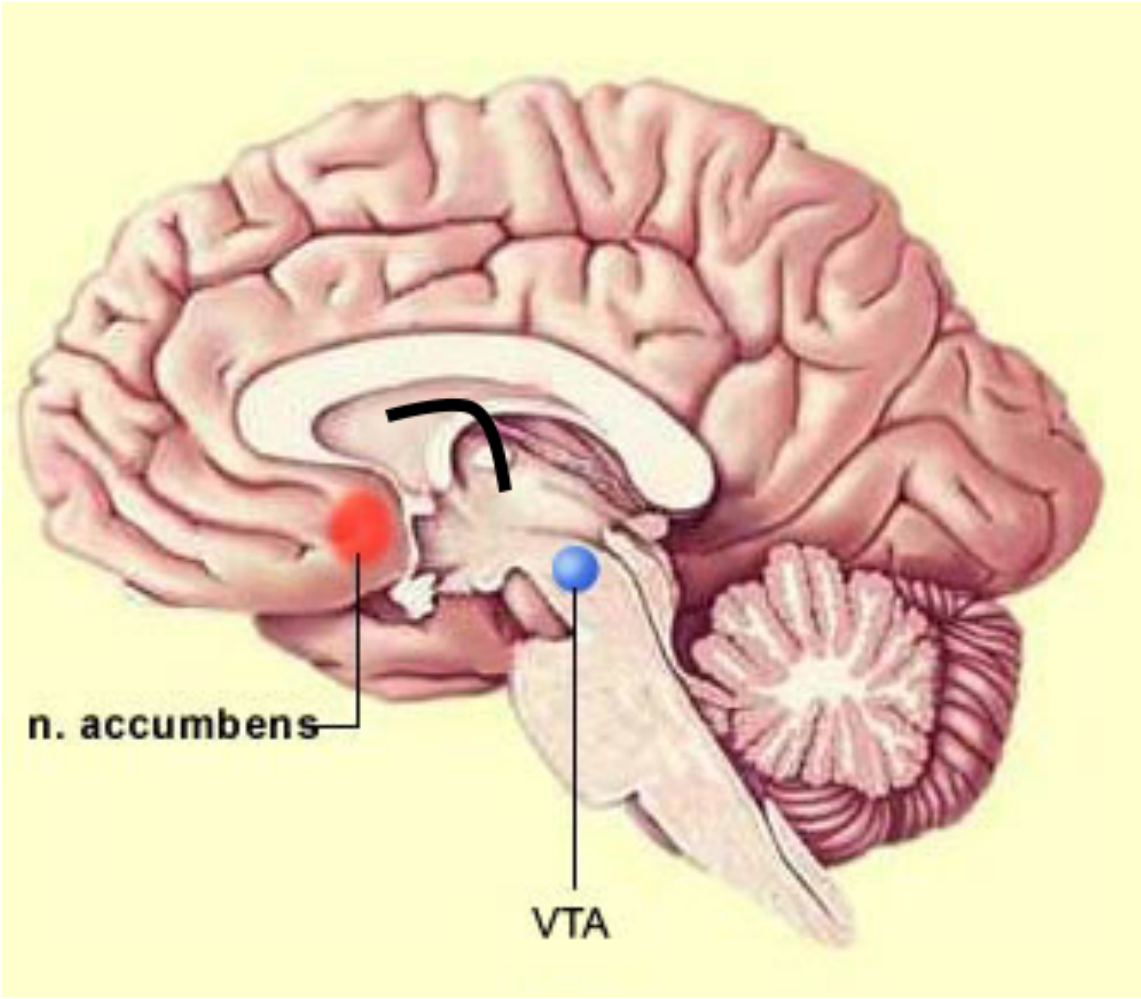
The weights w of the stimuli

The reward delivery system

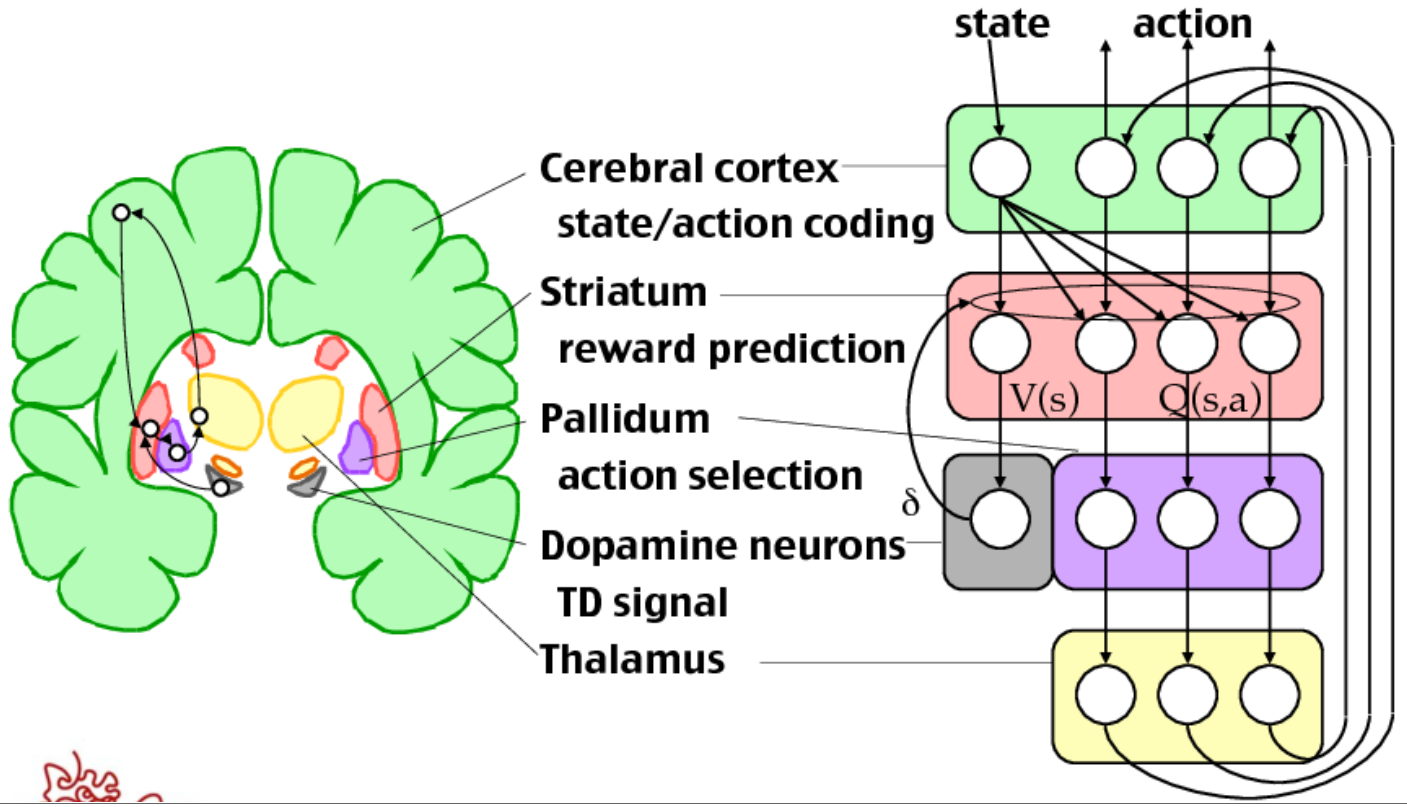
The reward circuit

Btw: the reward is

dopamine 



Reinforcement learning in the brain: we are still guessing the details…



RL: Summary

- In Pavlovian learning and its relatives, you learn from the stimuli
- In Reinforcement learning, you learn from your own actions
- In the brain dopamine and basal ganglia, striatum, VTA, nA, etc. seem to play a role in reinforcement learning
- (But we still do not know)
- Next: The math and deep learning of RL

Important model in RL: multi-armed bandits

m actions a = 1…m

Each has an unknown reward distribution Da

You are stuck at the casino for a very long time T

How would you play?

Warning: a highly dopaminergic problem

The multi armed bandits problem was formulated during the war, and efforts to solve it so sapped the energies and minds of Allied scientists that the suggestion was made that the problem be dropped over Germany, as the ultimate instrument of

Exploration vs. exploitation

- One machine has the best expectation, M*
- Best thing to do is play this forever – but you don't know which…
- So, at time t you choose A(t)
- Every machine i has a (unknown) gap Ga = M* - Ma
- You want to minimize regret: Regret(A) = Σt GA(t)

Exploration vs. exploitation

- You somehow start trying machines, evaluating their mean by sampling
- Greedy: Always stick to the best estimate, never exploit → regret is linear in T
- Noisy greedy: 10% of the time try a new machine, otherwise greedy → regret still linear in T

Exploration vs. exploitation: is linear regret inevitable?

- Lower bound: regret is at least log T · (Σa Ga/KL(a, a*))

Log regret is possible!

- Natural idea: if an action has good sample mean but few samples, punish it a little.
- But how much?
- Recall that the probability that sample mean is b away from true mean is about

$$\exp(-\text{samplesize} \cdot b2)$$

(Hoeffding bound)

Algorithm UCB1 [Auer, Cesa-Bianchi, and Fischer 2002]

Estimate the expectation of every action as

current mean + $\sqrt{(\log t/\text{sample size})}$,

- Note: punishment term encourages exploration of poorly sampled actions

- **Theorem**: UCB1 has regret at most $10 \log T \cdot (\Sigma a\ Ga)$

An even better algorithm: Posterior sampling

- Maintain a parametrized model (perhaps a deep net…) of the reward distributions: $p(r \mid a) = p(r, \theta a)$
- We sample rewards according to the current parameters
  Treat this sample as the truth and pick the best action
  Update the parameters ($\theta 1, \theta 2, \theta 3, \ldots$) according to the result
- This algorithm (Thompson's algorithm) is not only logarithmic, but it achieves the lower bound.

More advanced model: time-dependent bandits

Suppose now the machines are known Markov chains with a reward at each transition

We know the current states

Discounted rewards

Gittins index of a Markov chain

- Suppose you had a choice between a Markov chain M and the one-state Markov chain R (= reward)
- You are given the option to play M for a while and then R forever. For how long would you play M?
- Obviously, the larger R, the shorter.
- For large enough R, you will not touch M
- Gittins index of M: the smallest R for which you would not touch M

Gittins index of a Markov chain

Gittins Theorem: The best thing to do is always play the machine with the highest Gittins index

Markov decision processes (MDPs)

- Gittins allows you to switch Markov chains
- In MDPs the Markov chains all share the same states
- You have a choice of action at each state
- Your choice changes the transition probabilities and the rewards
- Strategy A: You choose, once and for all, what to do at each state, and then follow the resulting Markov chain
- What is the best strategy?

Example: the taxicab problem (From Ron Howard's book)

A taxi serves three adjacent towns: A, B, and C.

Each time the taxi discharges a passenger, the driver must

choose from three possible actions:

(1) "Cruise" the streets looking for a passenger.
(2) Go to the nearest taxi stand (hotel, train station, etc.)
(3) Wait for a radio call from the dispatcher with instructions (but not possible in town B because of distance and poor reception).

MDP model:

States: {A, B, C}

Action sets:

$K_A = \{1,2,3\}$, $K_B = \{1,2,3\}$, $K_C = \{1,2\}$

Transition probability matrices

Cruising streets	Waiting at taxi stand	Waiting for dispatch
$P^1 = \begin{bmatrix} 1/2 & 1/4 & 1/4 \\ 1/2 & 0 & 1/2 \\ 1/4 & 1/4 & 1/2 \end{bmatrix}$	$P^2 = \begin{bmatrix} 1/16 & 3/4 & 3/16 \\ 1/16 & 7/8 & 1/16 \\ 1/8 & 3/4 & 1/8 \end{bmatrix}$	$P^3 = \begin{bmatrix} 1/4 & 1/8 & 5/8 \\ 0 & 1 & 0 \\ 3/4 & 1/16 & 3/16 \end{bmatrix}$

$$R^1 = \begin{bmatrix} 10 & 4 & 8 \\ 14 & 0 & 18 \\ 10 & 2 & 8 \end{bmatrix} \qquad R^2 = \begin{bmatrix} 8 & 2 & 4 \\ 8 & 16 & 8 \\ 6 & 4 & 2 \end{bmatrix} \qquad R^3 = \begin{bmatrix} 4 & 6 & 4 \\ 0 & 0 & 0 \\ 4 & 0 & 8 \end{bmatrix}$$

Rewards

What is the best action at each city?

Value of a state: Bellman's equation

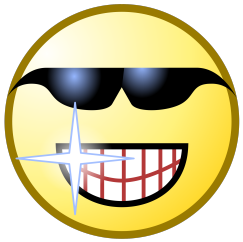
V[state] = maxA [R(state,A) + γ·EA [V[next state])

Note:

It's a linear program…

And you can iterate it

(Value iteration)



Another algorithm: Policy iteration

Start with any policy A

Calculate the values of each state

Policy improvement: update A using one-step look-ahead

(Can converge faster)

So, what's the problem?

- Problem is, Chess has 1050 states, Go has 10170 an automated driver may have more…
- Policy A is a parametrized function of the state
- Action at state s is $A(s,\theta)$
- E.g., θ is the set of weights in trained Alpha Go
- $J(\theta)$ = the expected reward of $A(s,\theta)$

How do you maximize $J(\theta)$?

- By policy iteration
- Which means, stochastic gradient ascent: calculate $\nabla\theta$ by sampling paths of the Markov chain (roll-outs)
- Problems:
  - Large variance, many samples needed
  - Fixed policy, samples may be repeating
  - Slow convergence, small learning constant

So, what to do?

Large variance, many samples needed

Simple solution: subtract from total reward of each sample a baseline equal to an estimate of the reward

Alternative: use value iteration…

Fixed policy, samples may be repeating

Use two different policies, one for evaluation and one for sampling paths (can work…)

Slow convergence, small learning constant

More tricks