Lecture 6: Computational Cognitive Modeling

Reinforcement Learning (pt. 3)

course website:
https://brendenlake.github.io/CCM-site/

Three levels of description *(David Marr, 1982)*

Computational

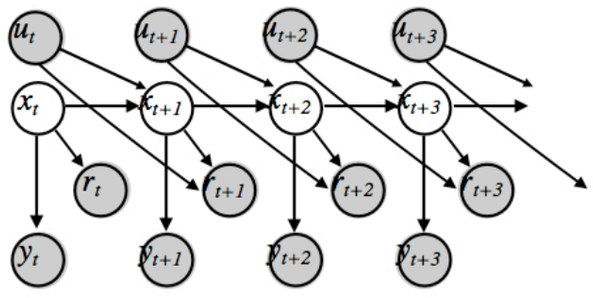
Why do things work the way they do?
What is the goal of the computation?
What are the unifying principles?

$$\textit{maximize:}\quad R_t = r_{t+1} + r_{t+2} + \cdots + r_T$$

Bellman

Algorthmic

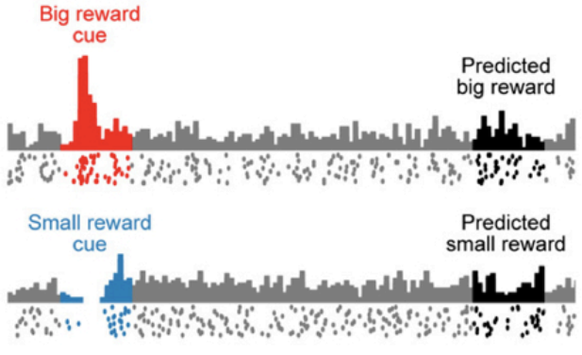
What representations can implement such computations?
How does the choice of representations determine the algorithm?



Dynamic programming, TD methods, Monte Carlo

Implementational

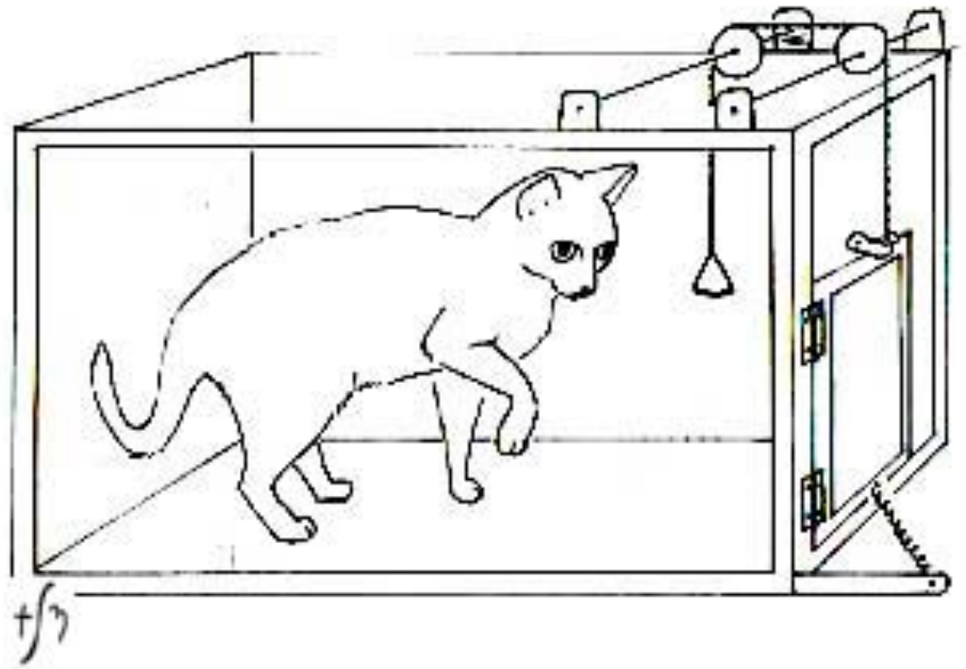
How can such a system be built in hardware?
How can neurons carry out the computations?



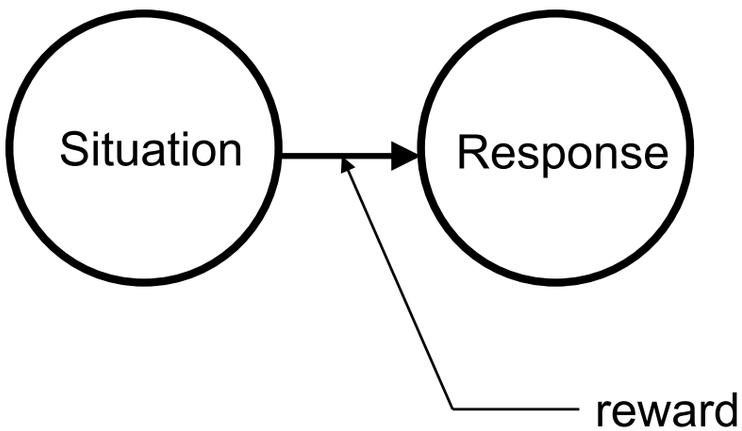
Neural firing patterns, prediction errors, system level neuroscience

Edward Thorndike

Law of Effect

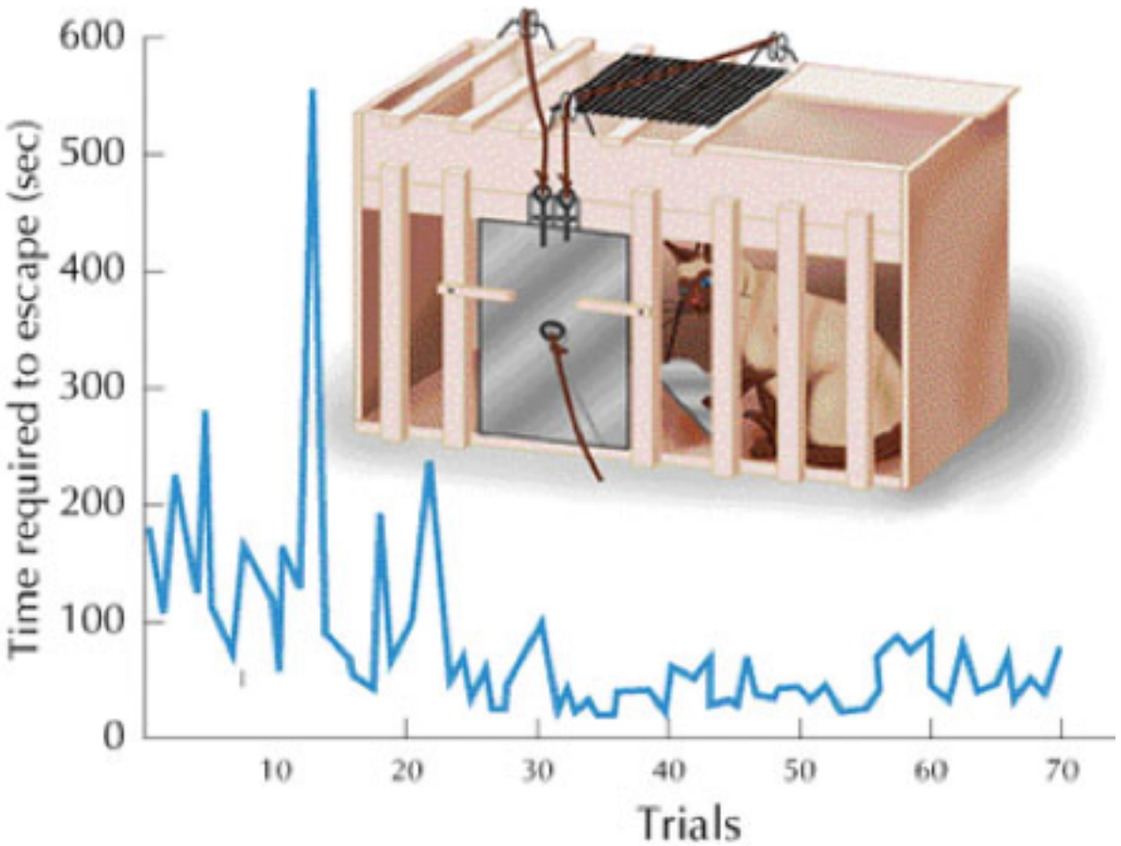


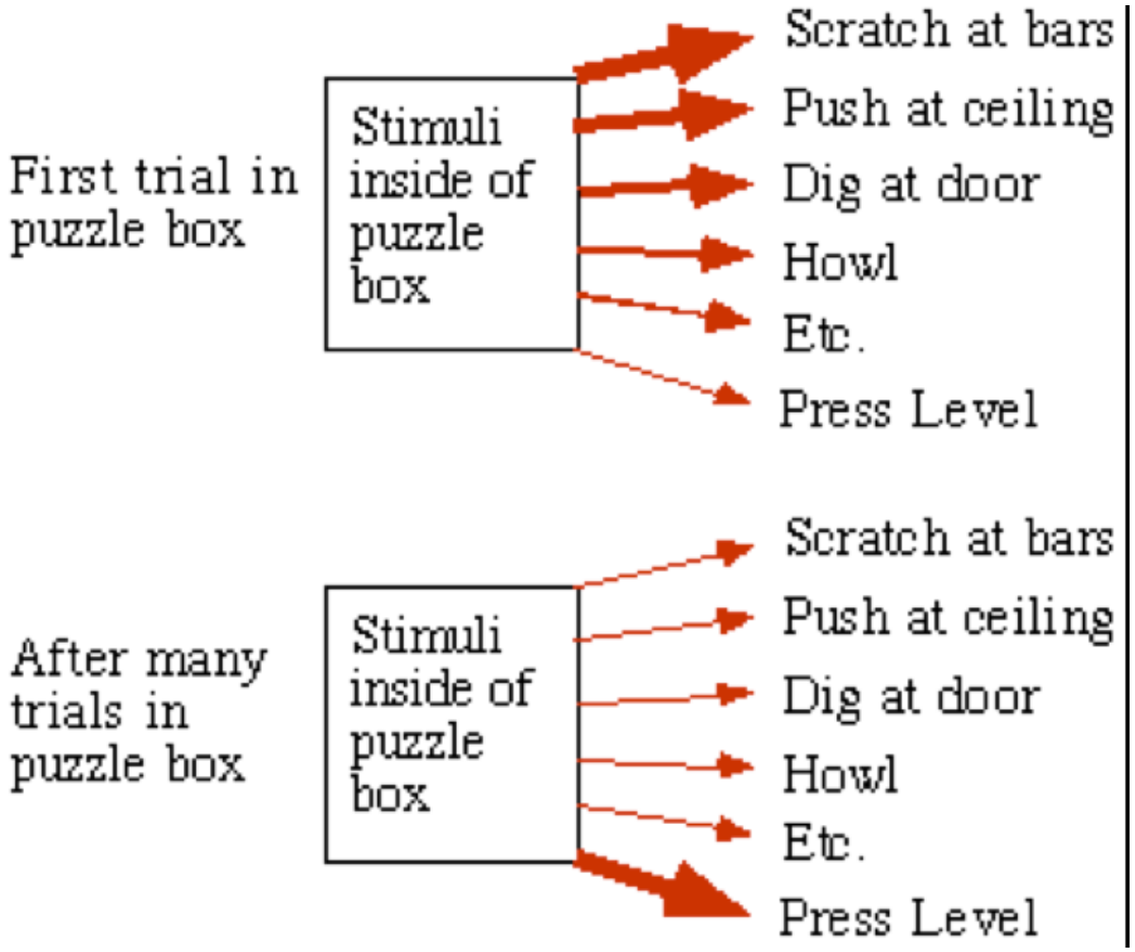
"Of several responses made to the same situation, those which are accompanied or closely **followed by satisfaction** to the animal **will**, other things being equal, **be more firmly connected with the situation**, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur" (Thorndike, 1911)



Edward Thorndike

Law of Effect





classical conditioning

- prediction
- … revealed by behavior
- … shaped by learning

Maximizing Reward

The learning algorithm which approximates the full Bellman Solution (temporal difference):

$$V_t = E[r_{t+1}] + V_{t+1}$$

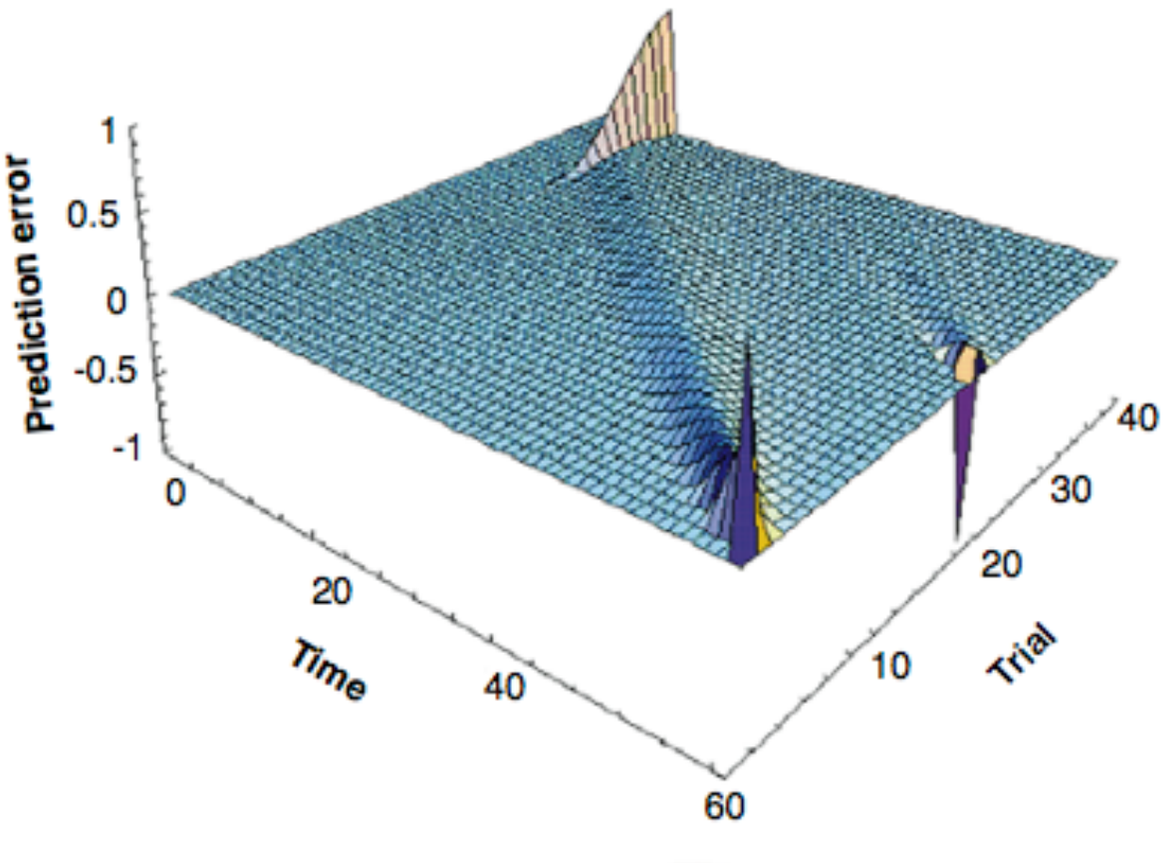
$$V_t \leftarrow V_t + \eta(r_{t+1} + V_{t+1} - V_t)$$

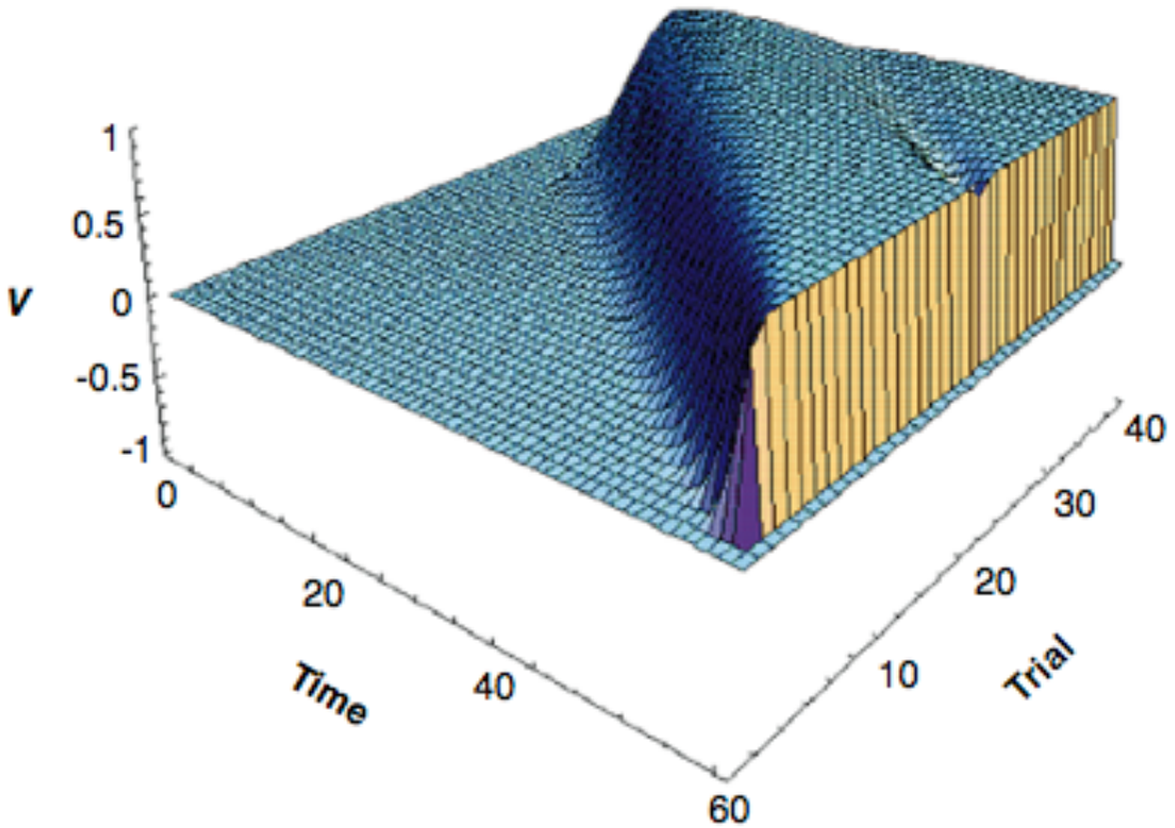
$$V_t \leftarrow (1-\eta)V_t + \eta(r_{t+1} + V_{t+1})$$

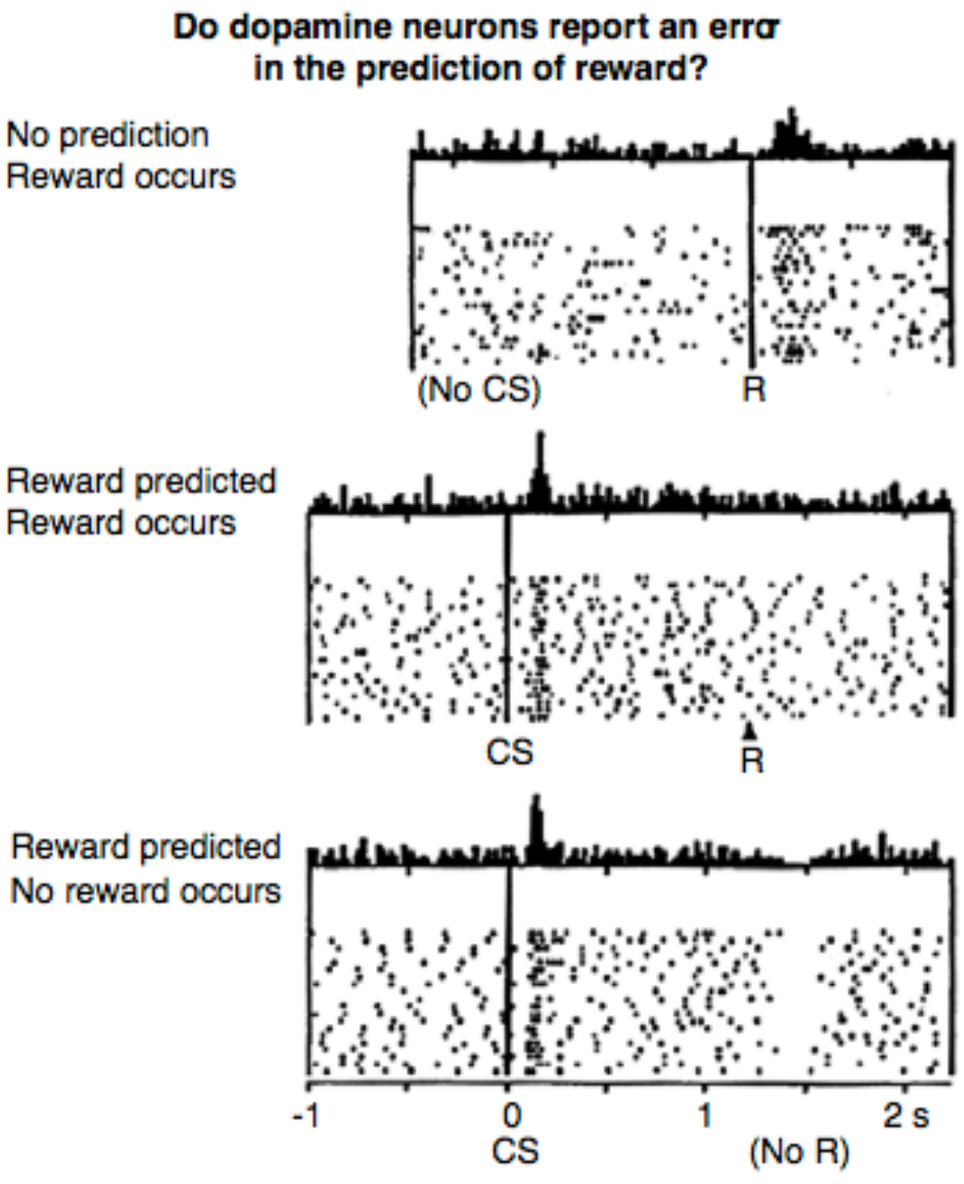
Compare with ***Rescorla-Wagner***

$$V_t \leftarrow V_t + \eta(r_{t+1} - V_t)$$

Temporal-difference learning and “backing up”

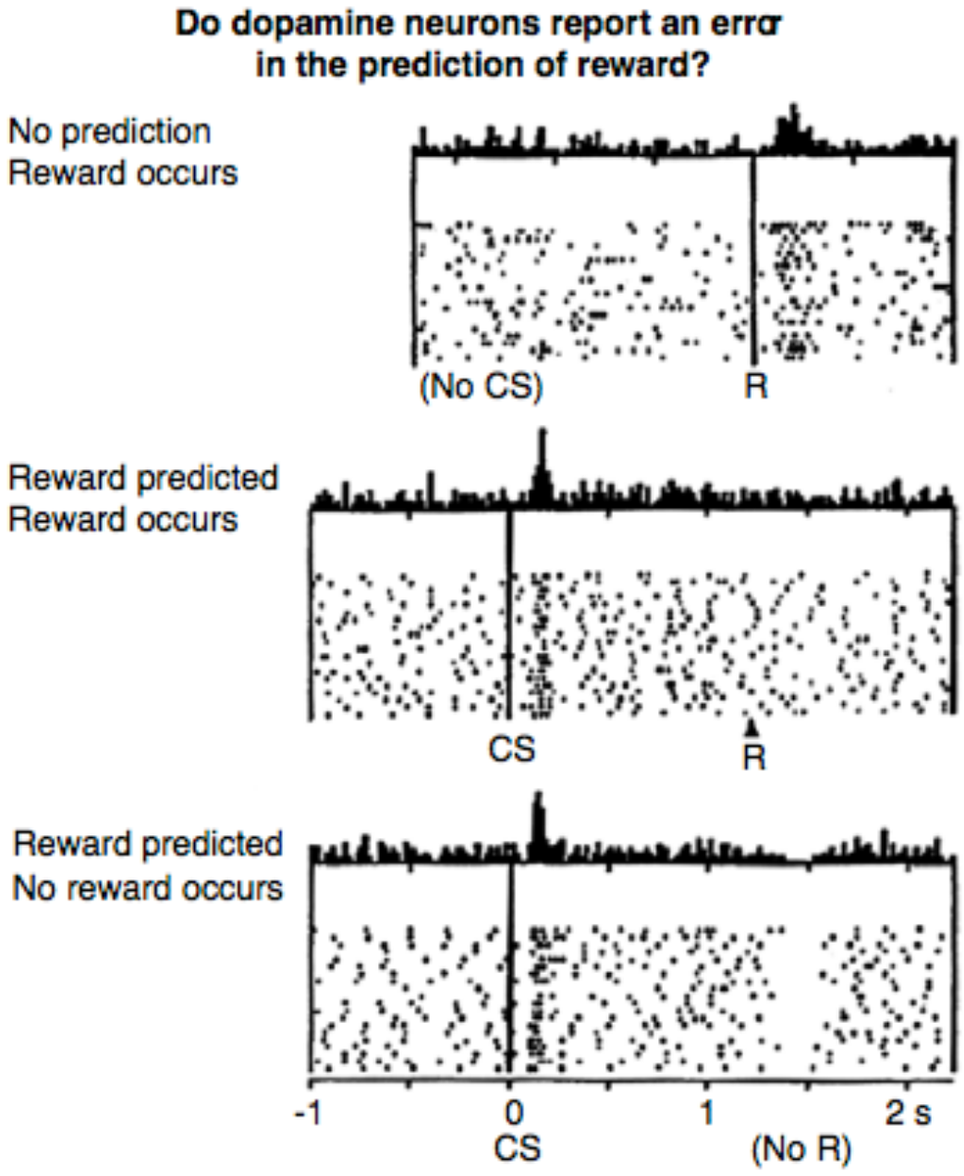






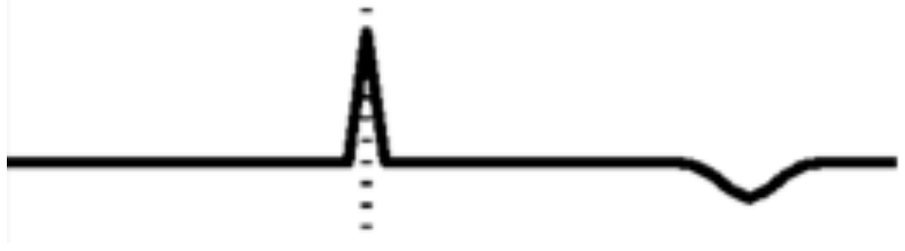
Dopamine neurons do it too...

Dopamine neurons do it too...



δ_t

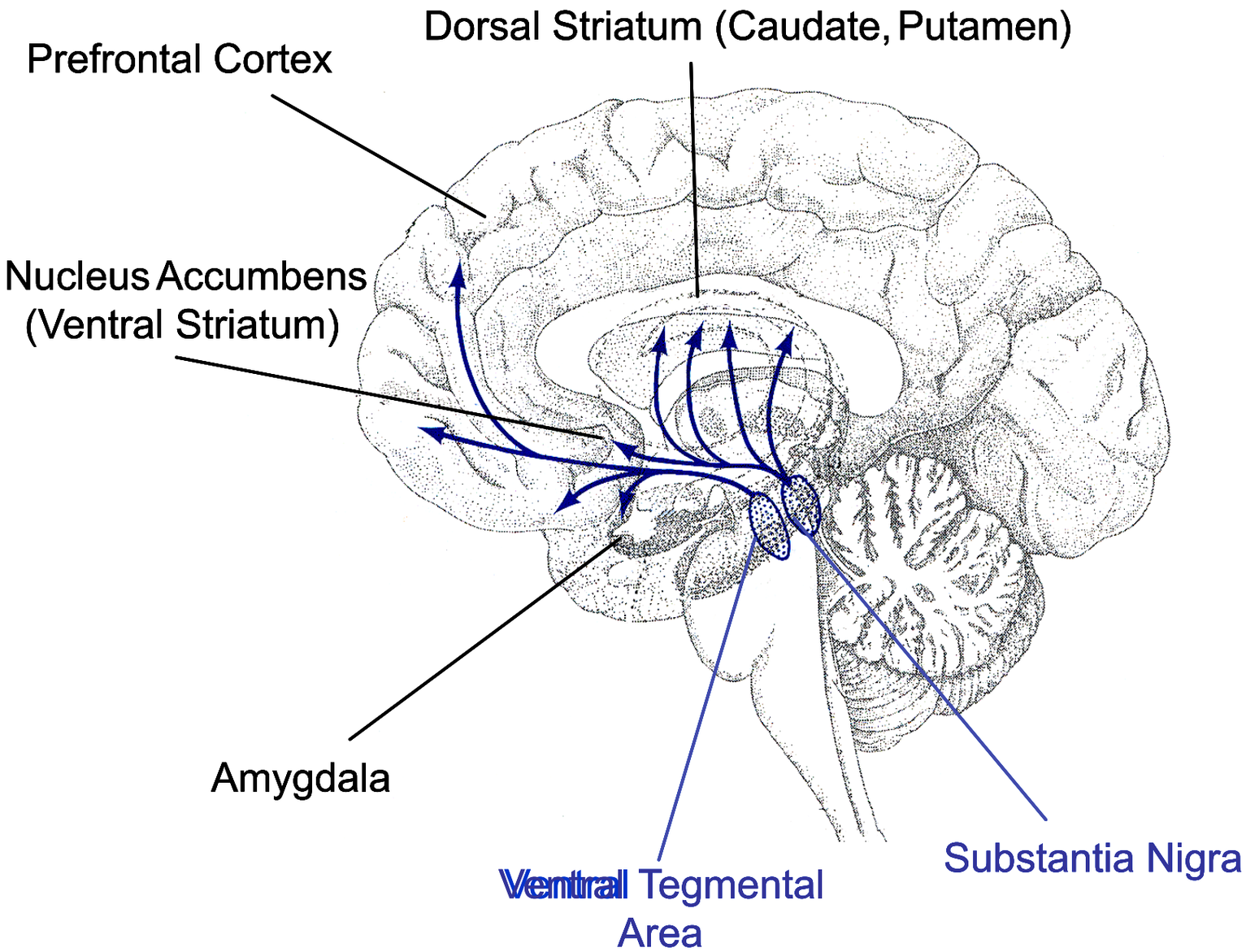
δ_t



δ_t

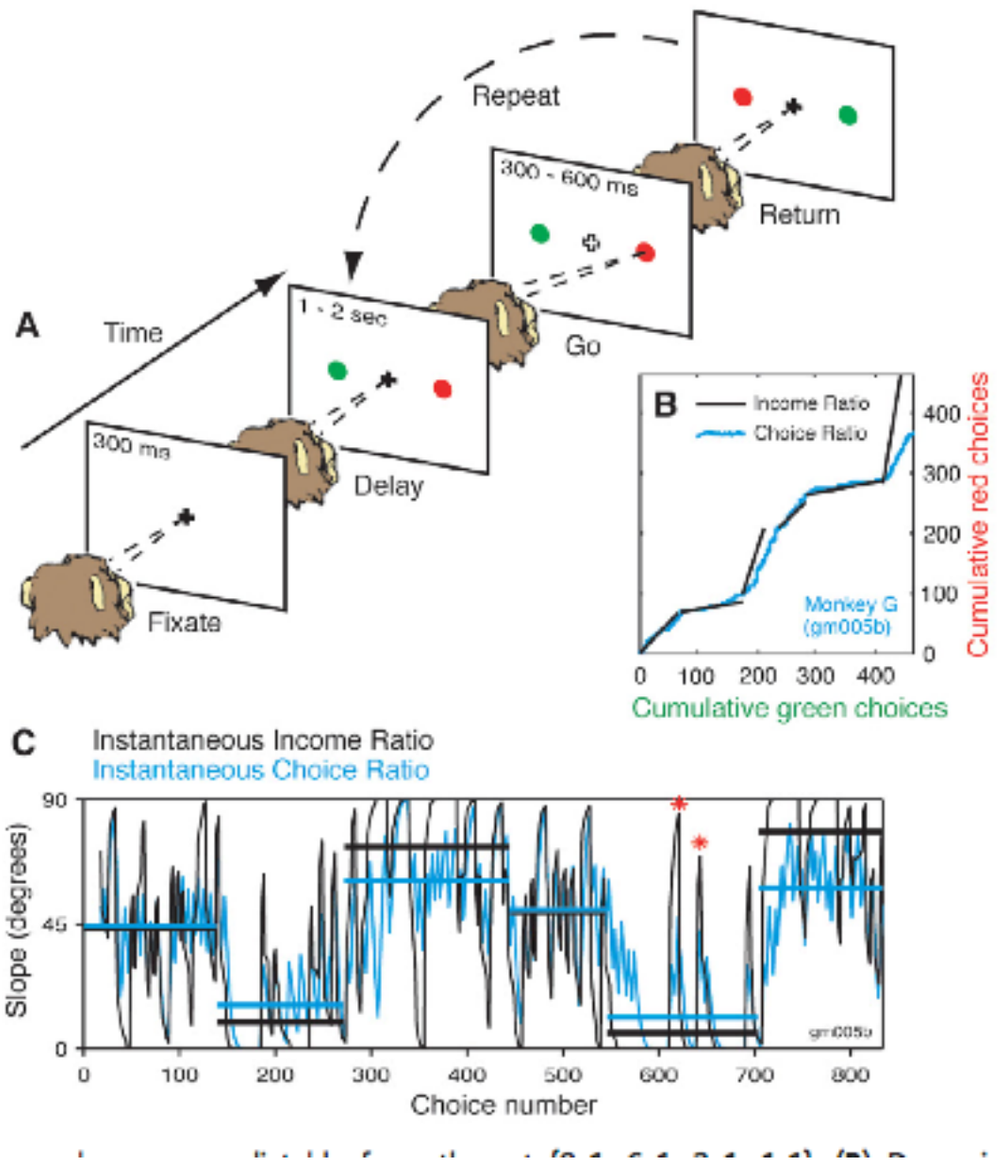
Also correlates with magnitude of reward, degree of intermittent reinforcement, etc...

The role of dopamine in reward learning



Matching Behavior and the Representation of Value in the Parietal Cortex

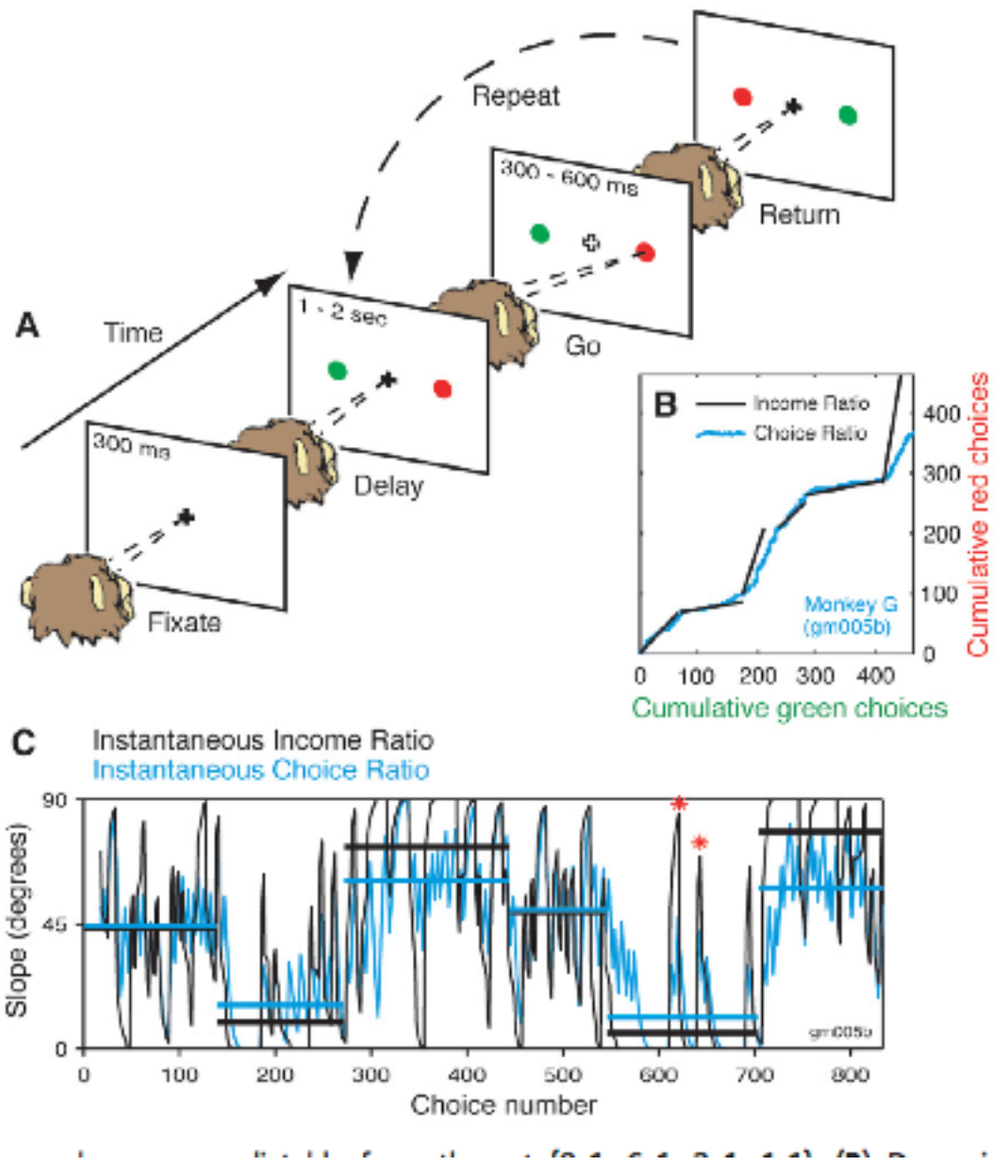
Leo P. Sugrue,* Greg S. Corrado, William T. Newsome



- Rhesus monkeys make eye fixations to one of two locations which get rewarded at different rates
- Dynamic foraging task since the reward from each option “drifts” in time (Poisson arrival times, but one are baited are always available... like VI schedule, makes matching the optimal strategy)
- Ever 200 or so trials, the reward magnitudes or rates change suddenly causing discontinuous shifts in the reward rates

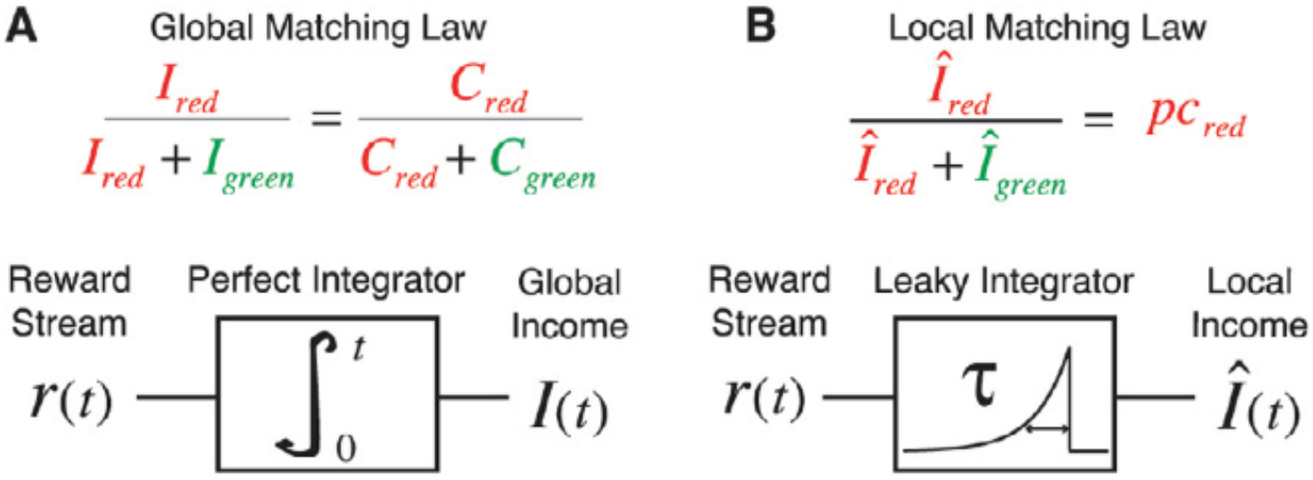
Matching Behavior and the Representation of Value in the Parietal Cortex

Leo P. Sugrue,* Greg S. Corrado, William T. Newsome



Learning

- Optimal integration of the reward history would be incapable of this quick adaptation
- Instead, suggest some “leaky integrator” which is pooling information in time (recency biased)

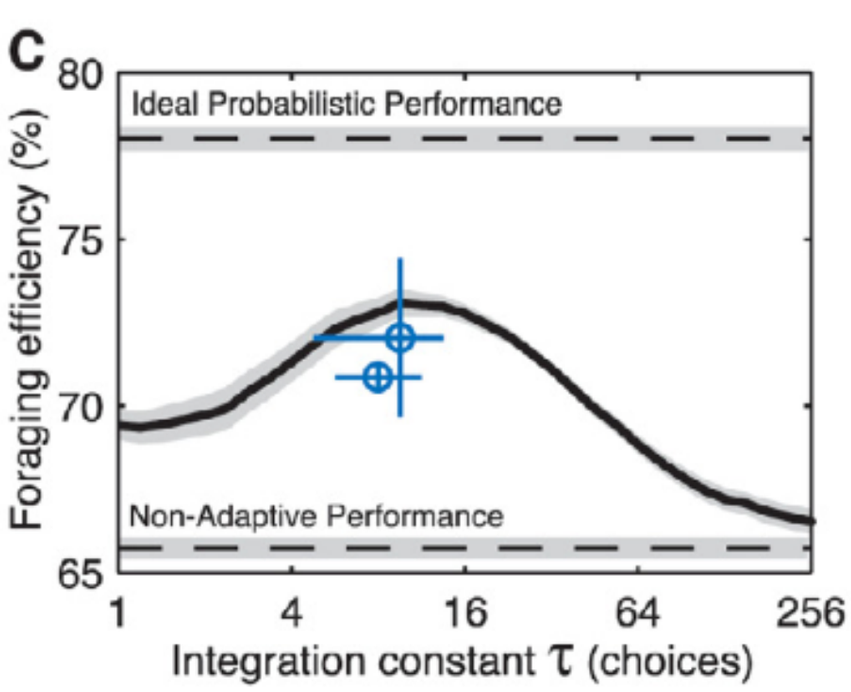


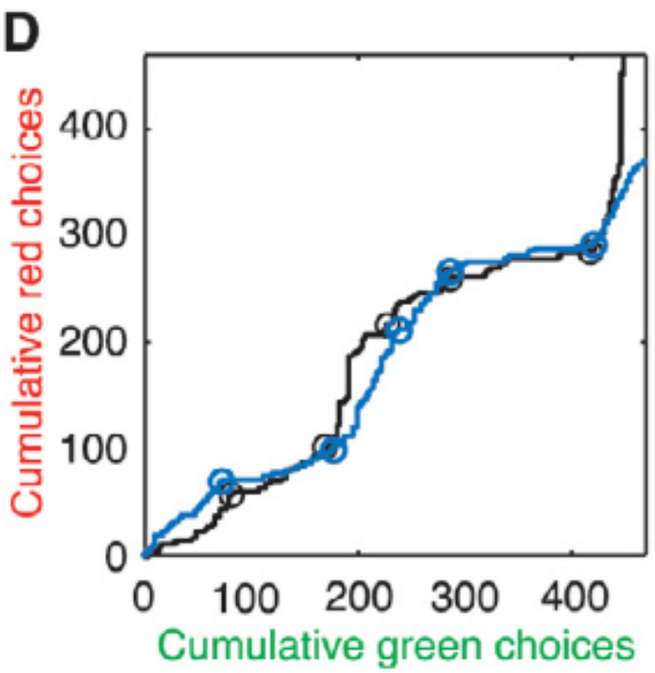
- Distinct from traditional matching law which integrates “globally”

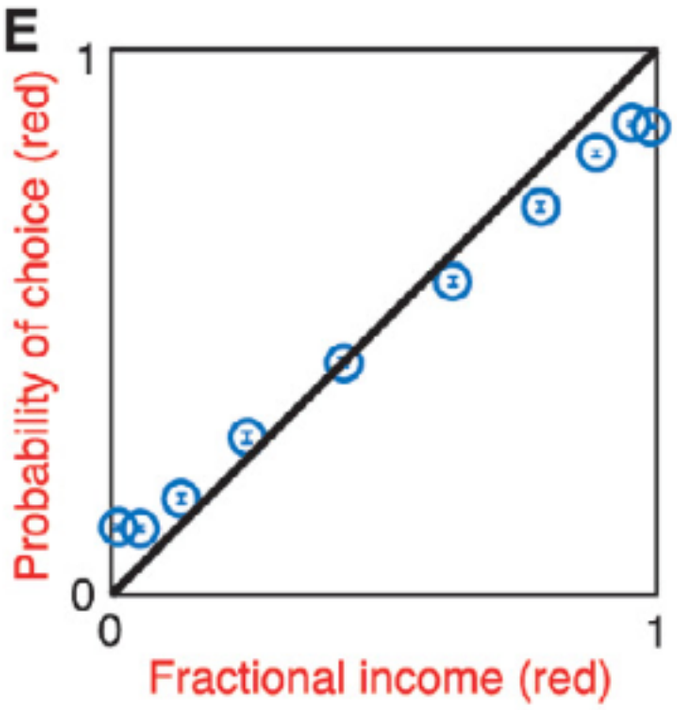
Matching Behavior and the Representation of Value in the Parietal Cortex

Leo P. Sugrue,* Greg S. Corrado, William T. Newsome

How do the monkeys do?







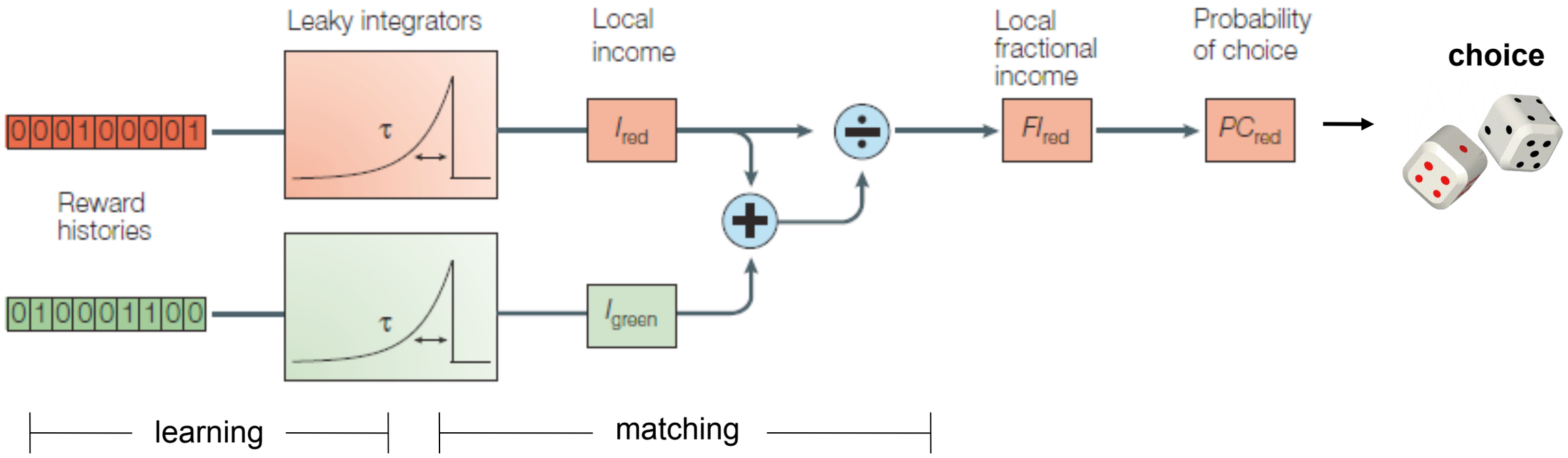
Near optimal setting of tau

The monkey's match!

Matching Behavior and the Representation of Value in the Parietal Cortex

Leo P. Sugrue,* Greg S. Corrado, William T. Newsome

The model

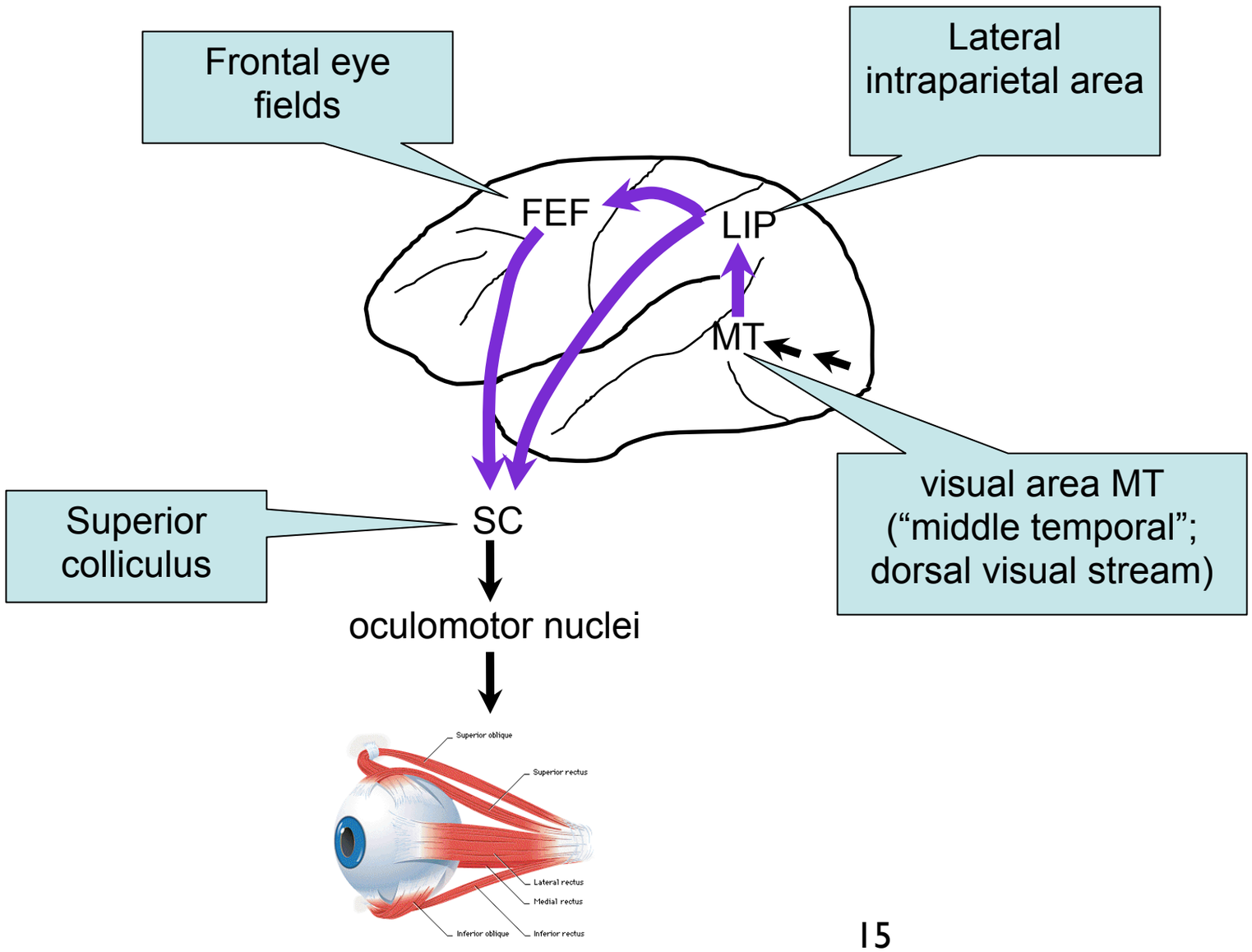


Matching Behavior and the Representation of Value in the Parietal Cortex

Leo P. Sugrue,* Greg S. Corrado, William T. Newsome

Recording from target-selective neuron is LIP

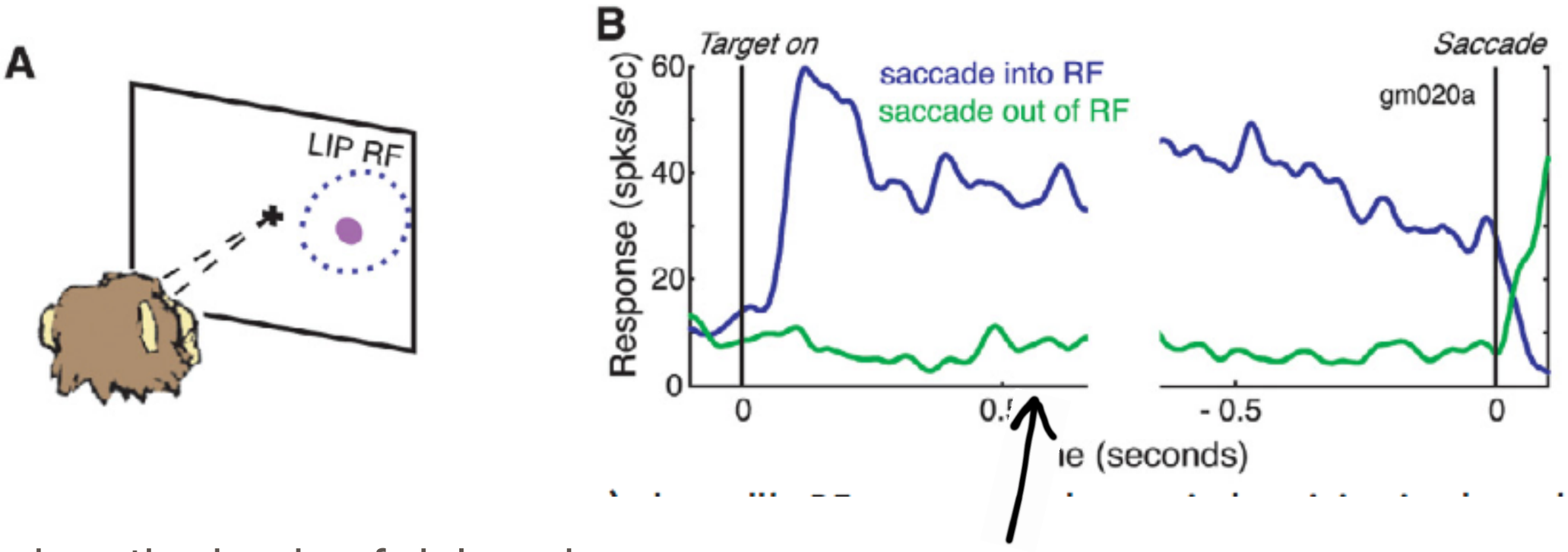
- Region involved in planning and executing saccades



Matching Behavior and the Representation of Value in the Parietal Cortex

Leo P. Sugrue,* Greg S. Corrado, William T. Newsome

Recording from target-selective neuron is LIP



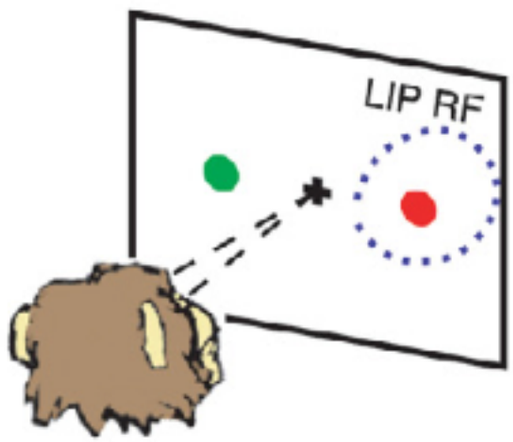
- Selected on the basis of delayed saccade task to find RF selective cells (like a forced choice version of matching game)

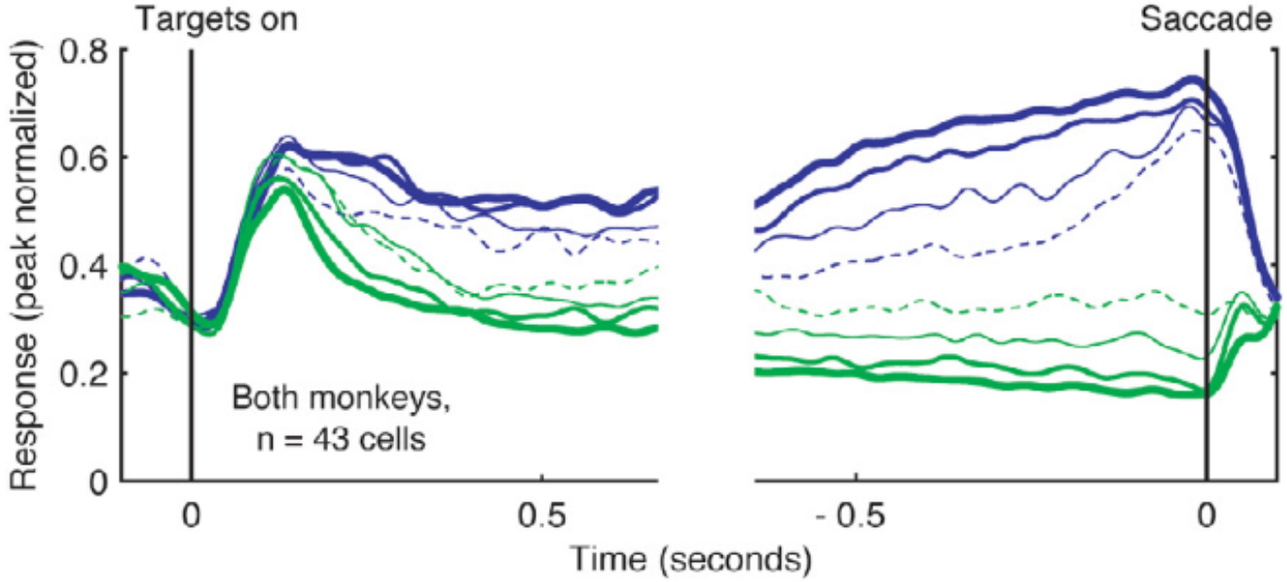
Spatially selective response

Matching Behavior and the Representation of Value in the Parietal Cortex

Leo P. Sugrue,* Greg S. Corrado, William T. Newsome

Recording from target-selective neuron in LIP



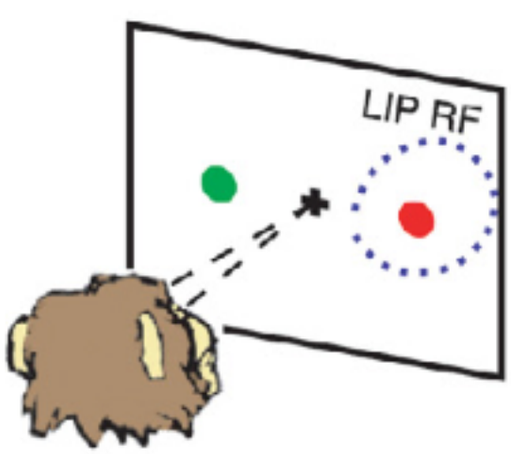


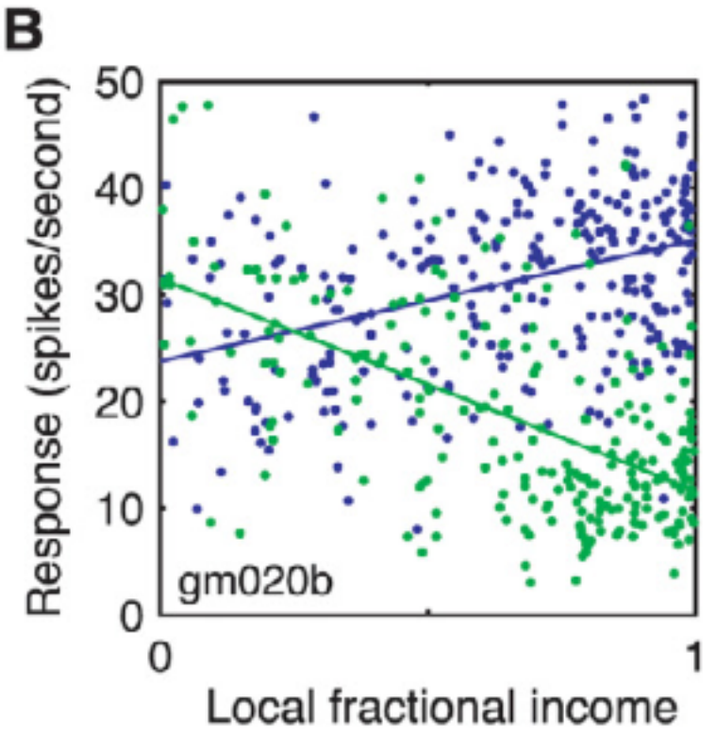
- Cells modulated by trial-to-trial utility of target in the receptive field

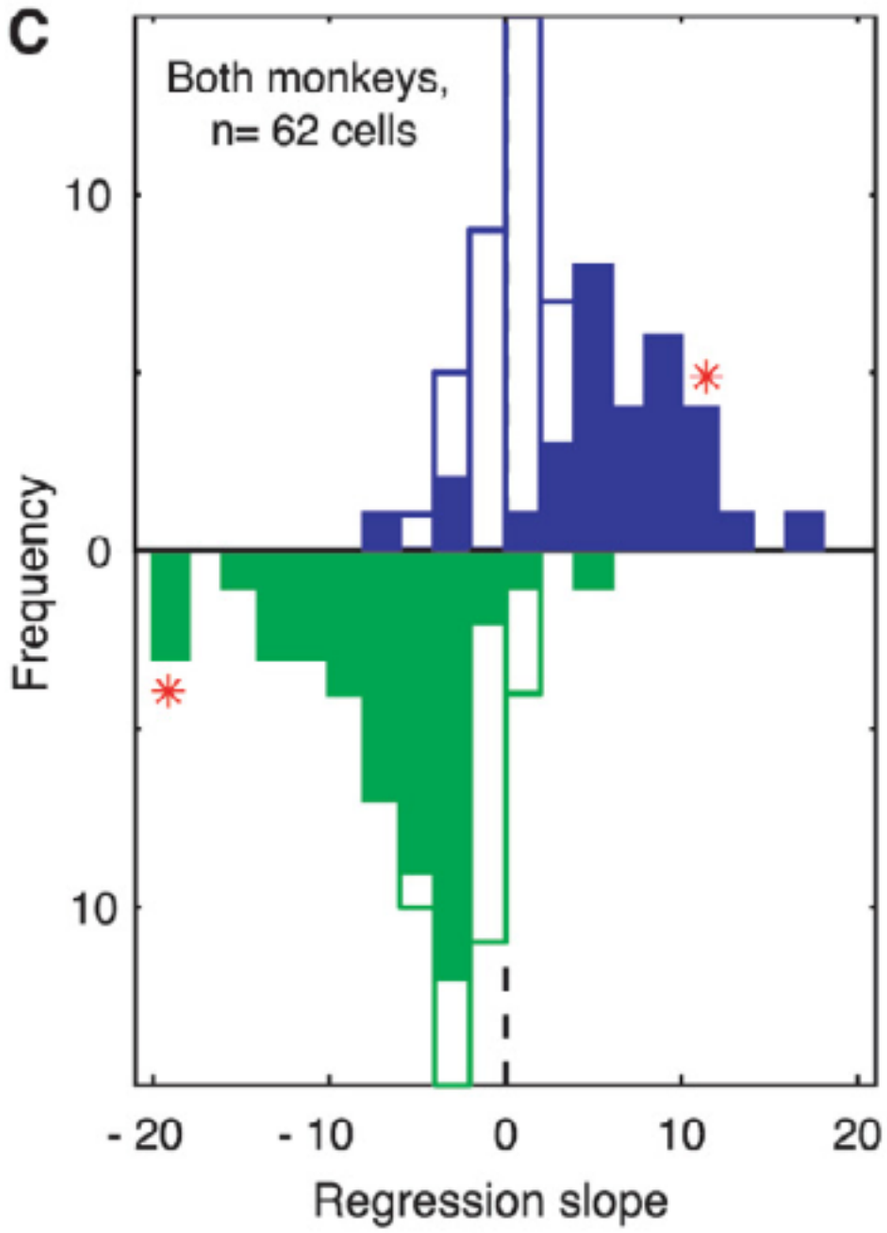
Matching Behavior and the Representation of Value in the Parietal Cortex

Leo P. Sugrue,* Greg S. Corrado, William T. Newsome

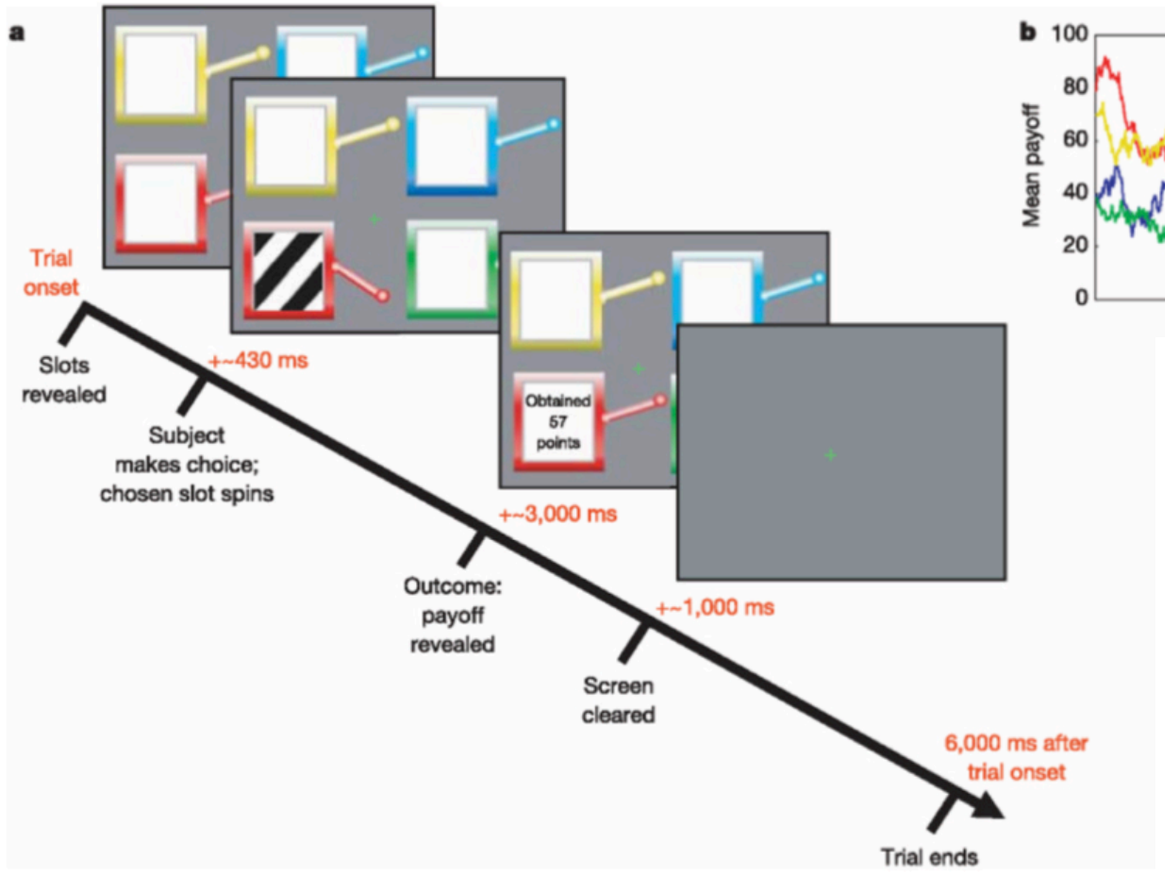
Recording from target-selective neuron is LIP







- Cells modulated by trial-to-trial utility of target in the receptive field



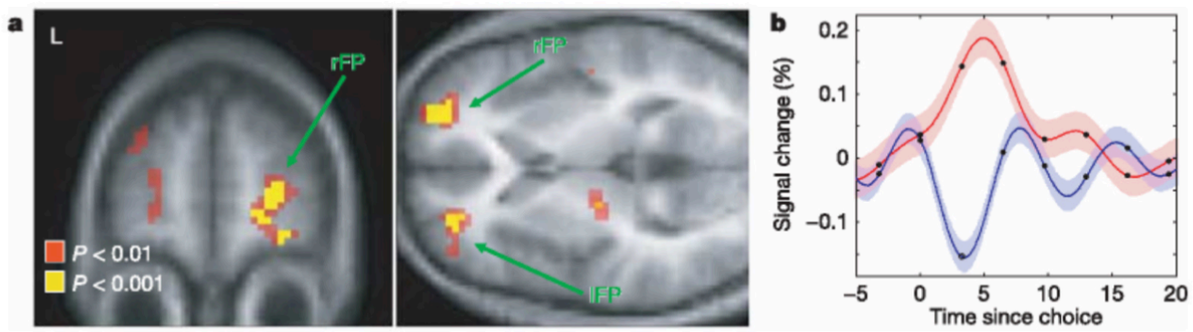


Figure 3.
Exploration-related activity in frontopolar cortex. a, Regions of left and right frontopolar cortex (lFP, rFP) showing significantly increased activation on exploratory compared with exploitative trials. Activation maps (yellow, $P < 0.001$; red, $P < 0.01$) are superimposed on a subject-averaged structural scan. The coordinates of activated areas are [−27,48,4, peak $z = 3.49$] for lFP and [27,57,6, peak $z = 4.13$] for rFP. b, rFP BOLD time courses averaged over 1,515 exploratory (red line) and 2,646 exploitative (blue line) decisions. Black dots indicate the sampling frequency (although, because sample alignment varied from trial to trial, time courses were upsampled). Coloured fringes show error bars (representing s.e.m.).

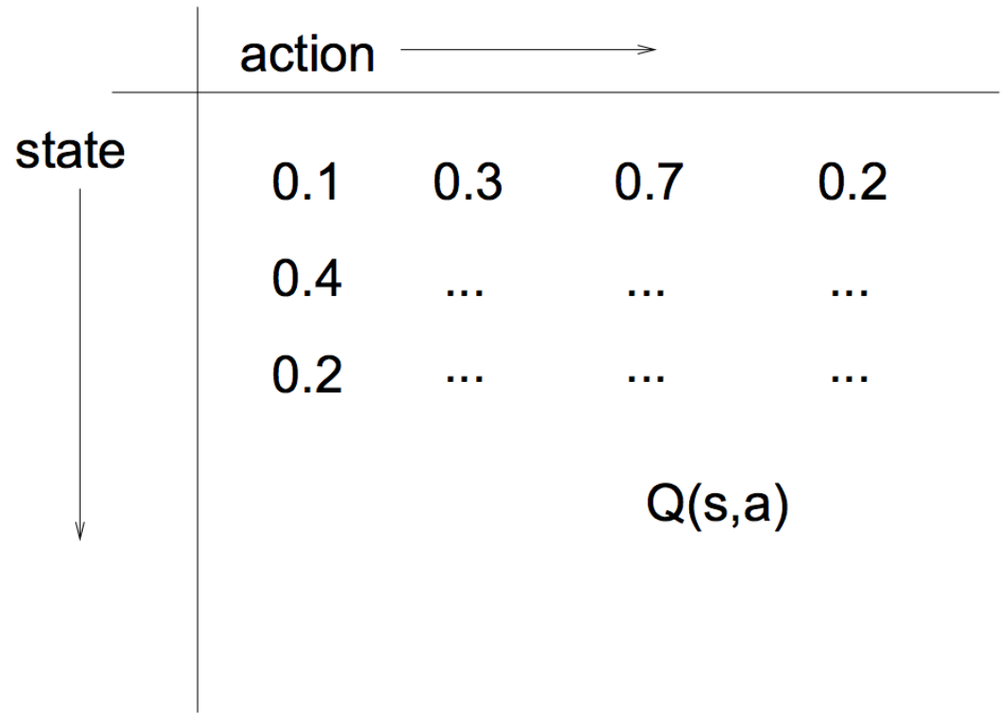
Published in final edited form as:
Nature. 2006 June 15; 441(7095): 876–879. doi:10.1038/nature04766.

Cortical substrates for exploratory decisions in humans

Nathaniel D. Daw, **John P. O'Doherty**, **Peter Dayan**, **Ben Seymour**, and **Raymond J. Dolan**

Generalization and Function Approximation

- Tabular value functions



What happens if the size of the state/action space is large?

- Large numbers of states/actions?
- Continuously-valued states/actions?
- Most states never experienced exactly before

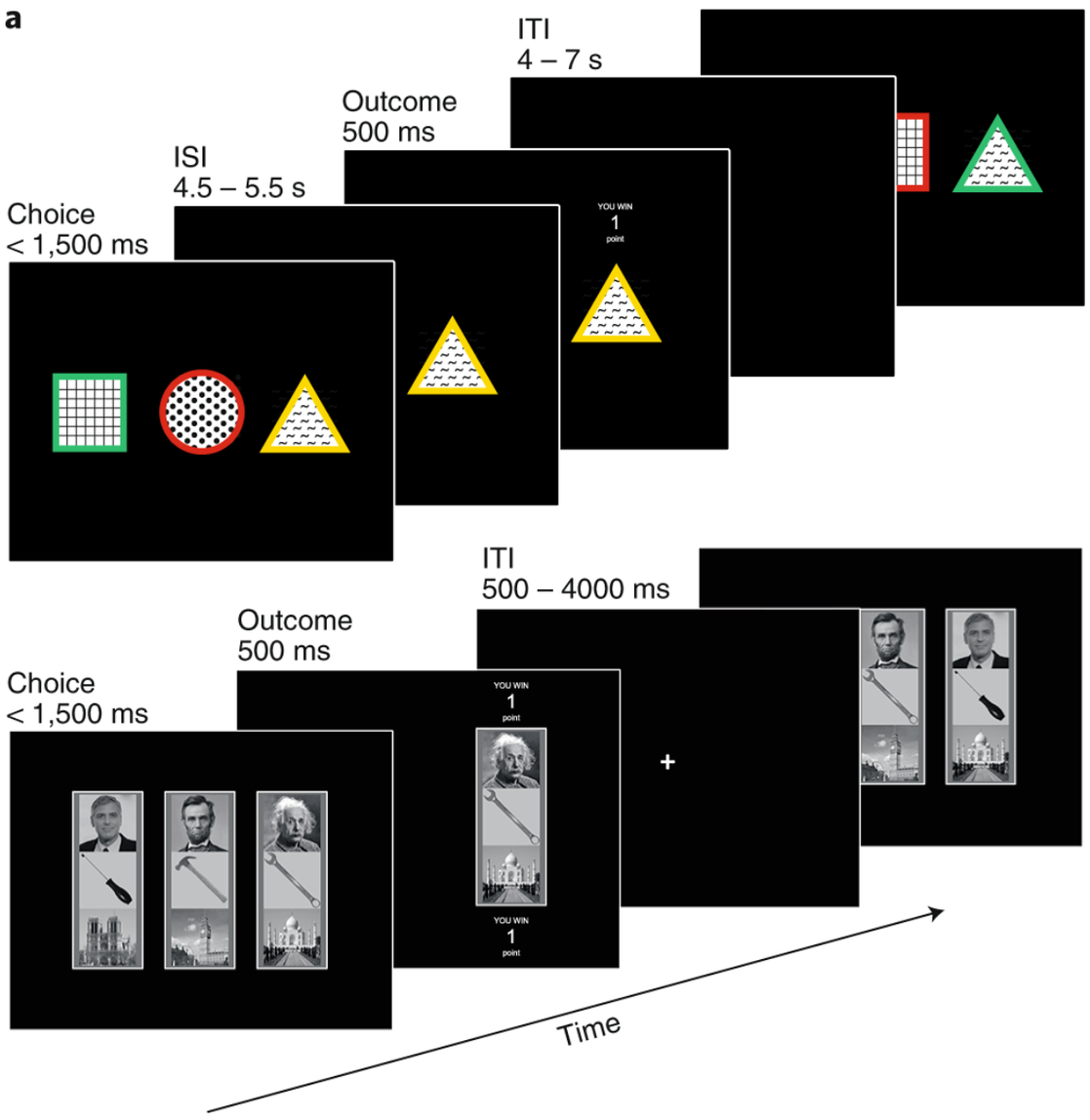
  Memory
  Time
  Data

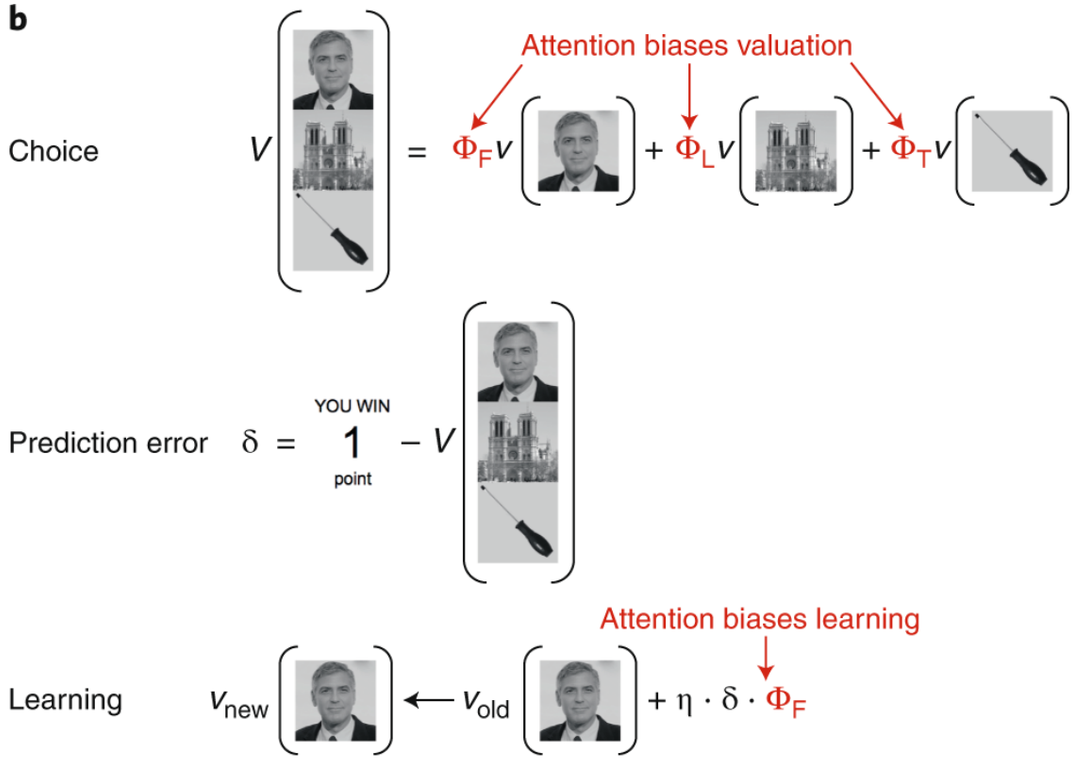
GENERALISATION: how experience with small part of state space is used to produce good behaviour over large part of state space

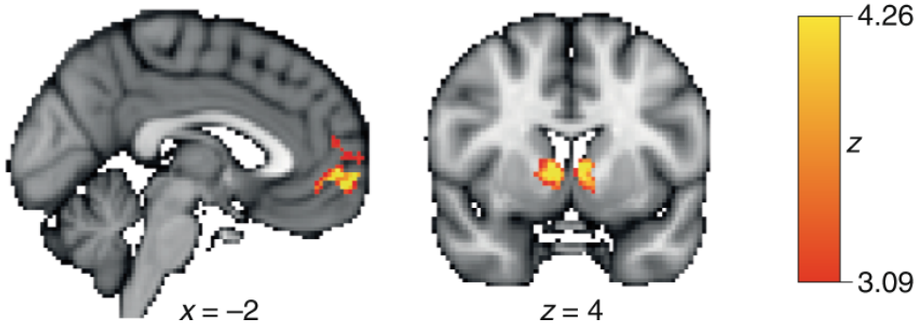
PERSPECTIVE | FOCUS
https://doi.org/10.1038/s41593-019-0470-8
nature neuroscience
https://doi.org/10.1038/s41593-019-0470-8

Learning task-state representations

Yael Niv





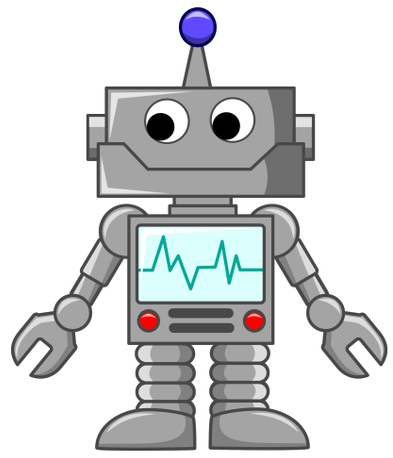


(Niv, 2019)

Generalization and Function Approximation

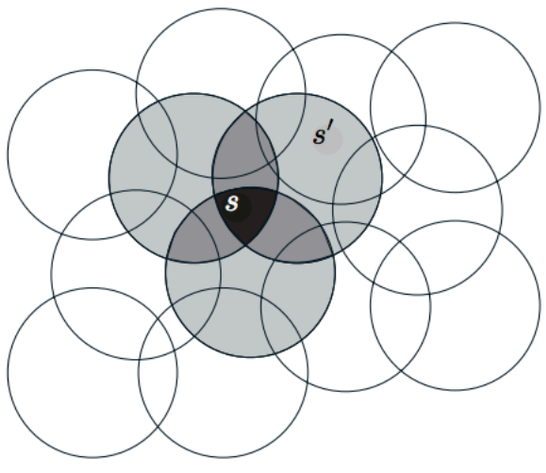
- Instead of directly looking up the states in a kind of addressable-memory, we represent different features of the the states

$$\vec{\phi}_s = \begin{pmatrix} x \\ y \\ \text{heading} \\ \text{batterypower} \end{pmatrix}$$



- Position in x, y coordinates (real numbers)
- Heading in degrees w.r.t. north (real number or quantised)
- Battery power = some real number

Coarse coding/tiling



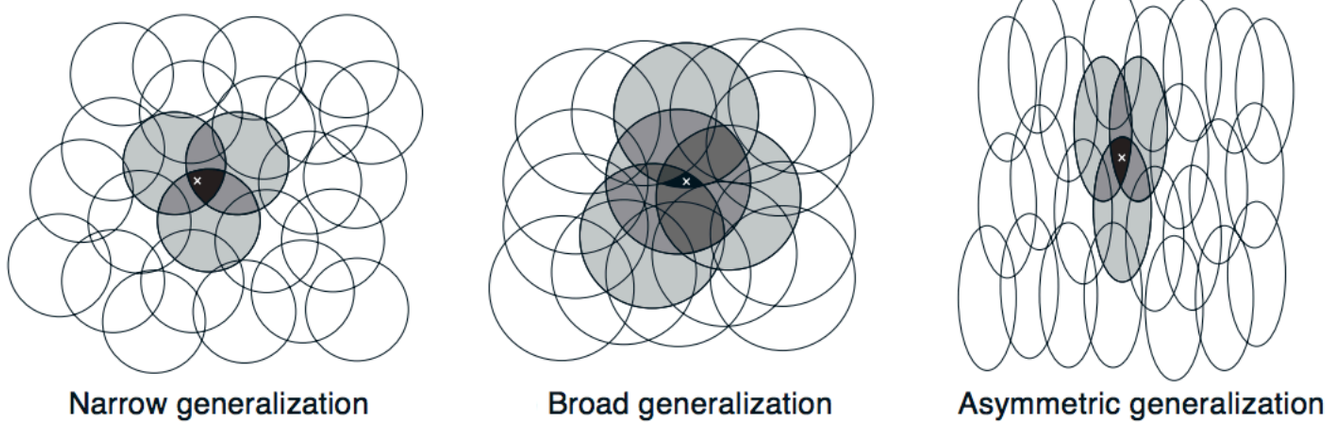


Figure 9.7: Generalization in linear function approximation methods is determined by the sizes and shapes of the features' receptive fields. All three of these cases have roughly the same number and density of features.

Generalization and Function Approximation

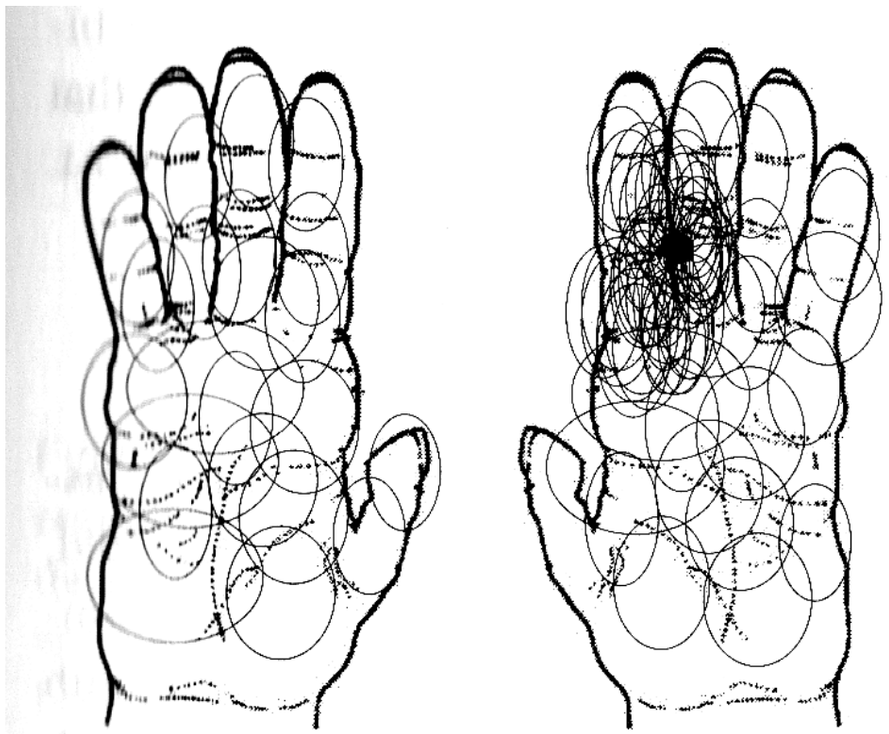
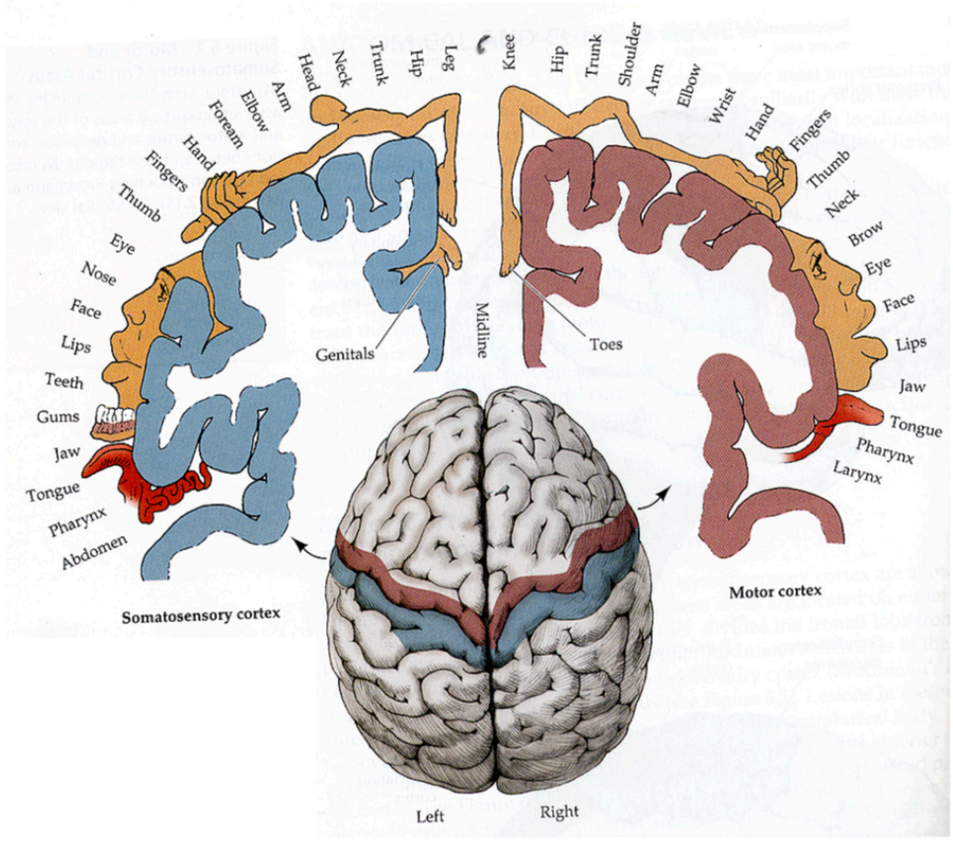


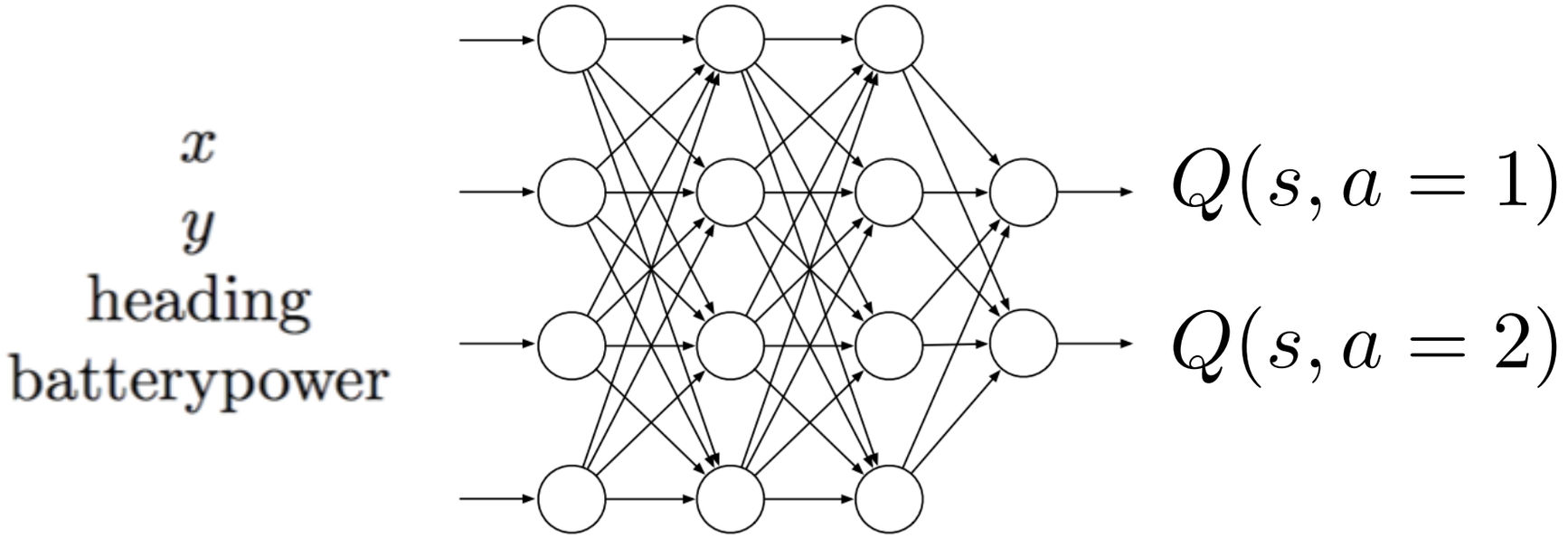
Figure 3.6 — CORTICAL PLASTICITY. The density of receptive fields of somatosensory cortical neurons, mapped onto the monkey palm surface, after prolonged exposure to local stimulation (*right*; the arrow indicates the stimulated spot). The map for the unstimulated hand is shown as a control (*left*). Merely touching the skin repeatedly at the designated location for the period of several days caused that location to become over-represented in the brain.



More neuron (finer tiling) = higher fidelity, better discrimination

Generalization and Function Approximation

- Neural networks make a useful approximation scheme due to the universal approximation properties



- Train with backprop as in past stuff but the loss function is the Q-learning update rule

$$L = \frac{1}{2}[r + max_{a'}Q(s', a') - Q(s, a)]^2$$

Reminder: Backpropagation algorithm for computing gradient

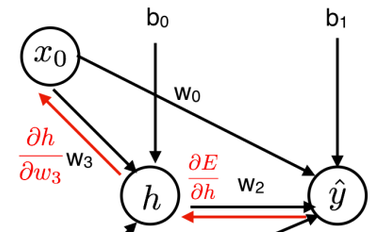
$$E(w, b) = (\hat{y} - y)^2$$
$$= (g(\text{net}) - y)^2$$

Multi-step strategy:

$$\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial h}\frac{\partial h}{\partial w_3}$$

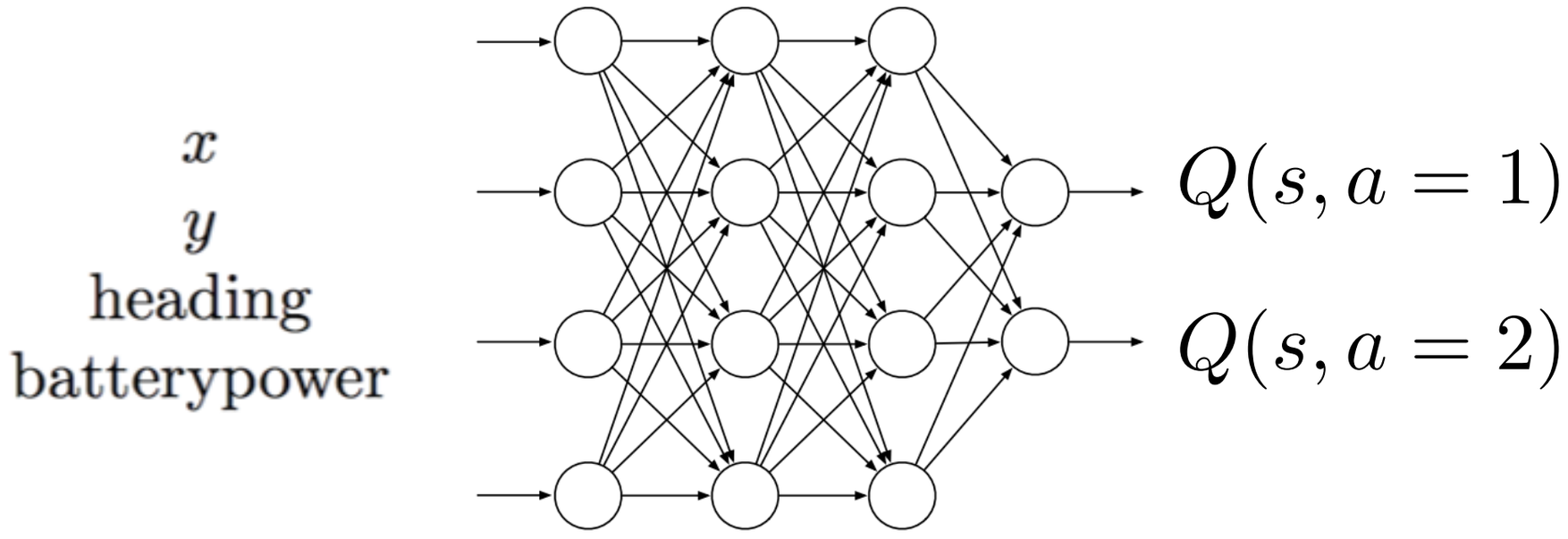
Step 1) Compute how error changes as a function of hidden unit activation

Step 2) Compute how hidden unit activation changes as a function of weight

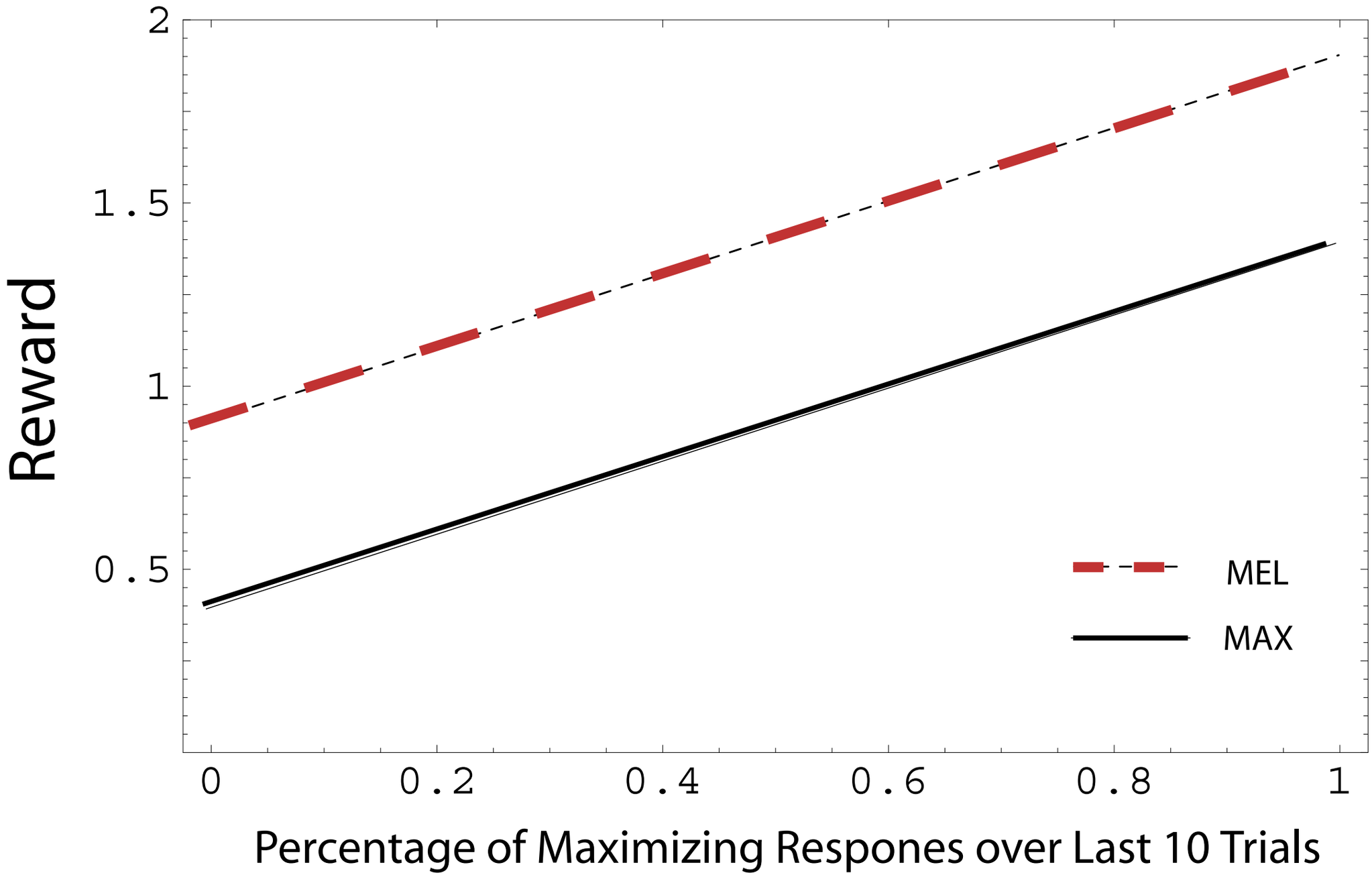


Q-learning gradient updates

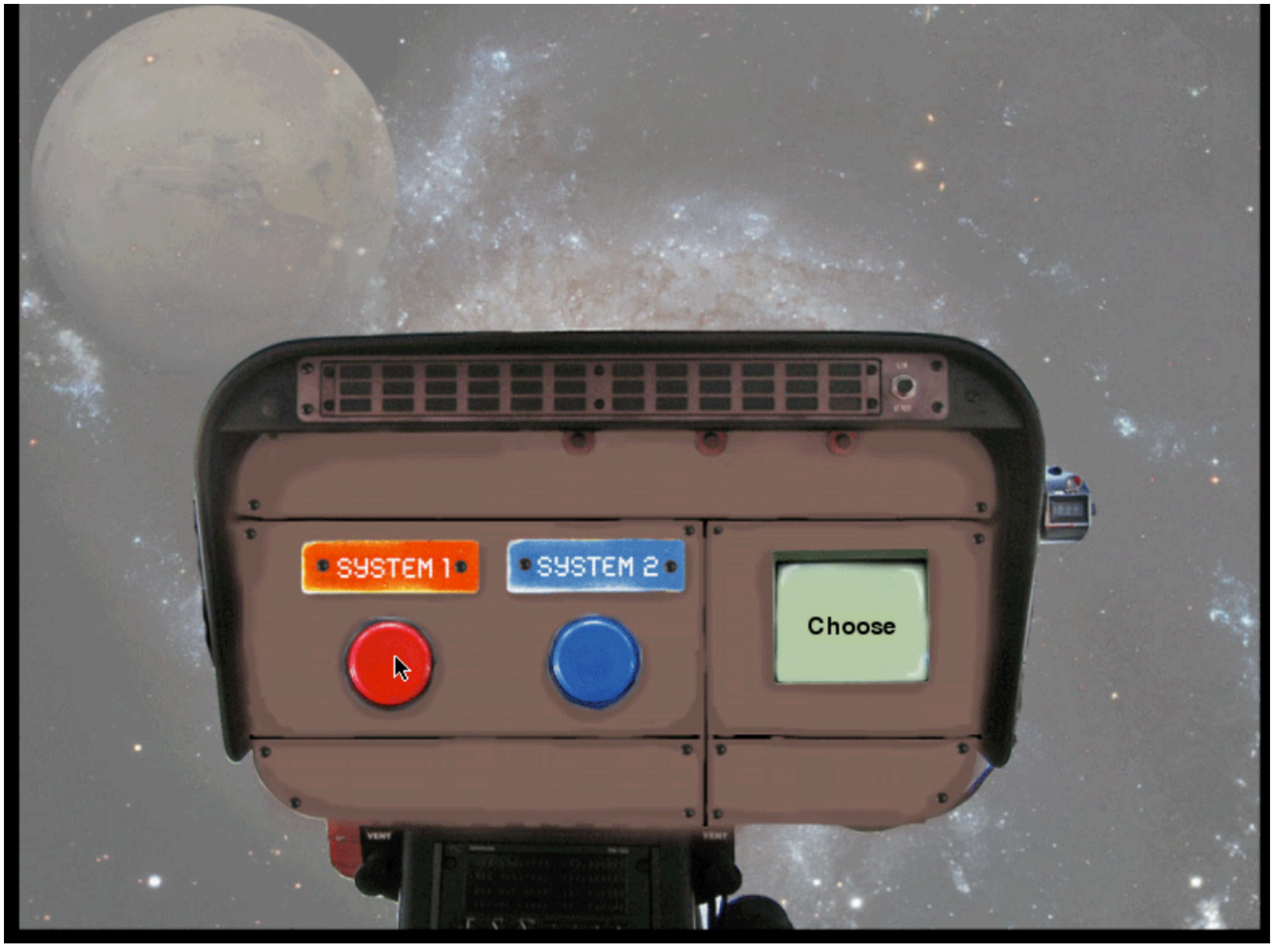
- Do a feedforward pass of the network for the current state s and get predicted Q values for all actions (output nodes)
- Choose an action using e.g., softmax of epilson greedy to encourage exploration
- Do a feedforward pass for the network for the next state (s') and get the maximum possible Q-value
- Set Q-value target for the chose action to $r + max_{a'}Q(s', a')$ for the chose action and zero for all others
- Backpropogate that error as for normal network

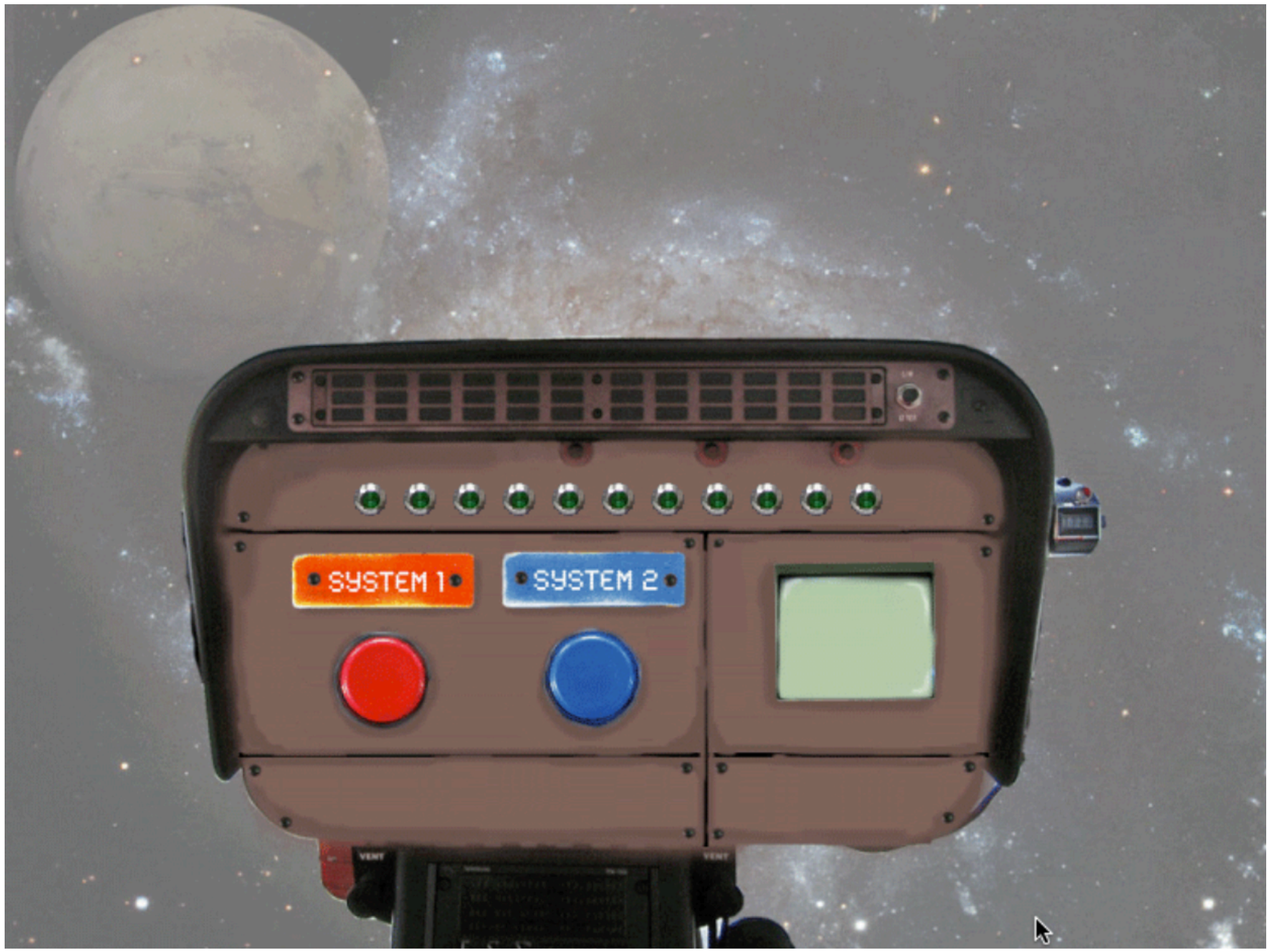


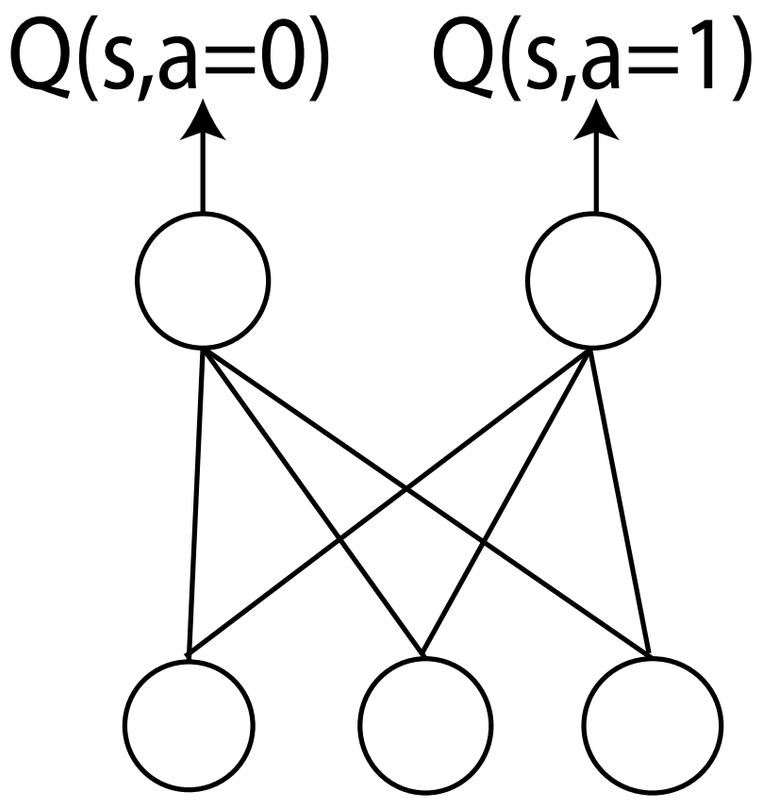
$$L = \frac{1}{2}[r + max_{a'}Q(s', a') - Q(s, a)]^2$$

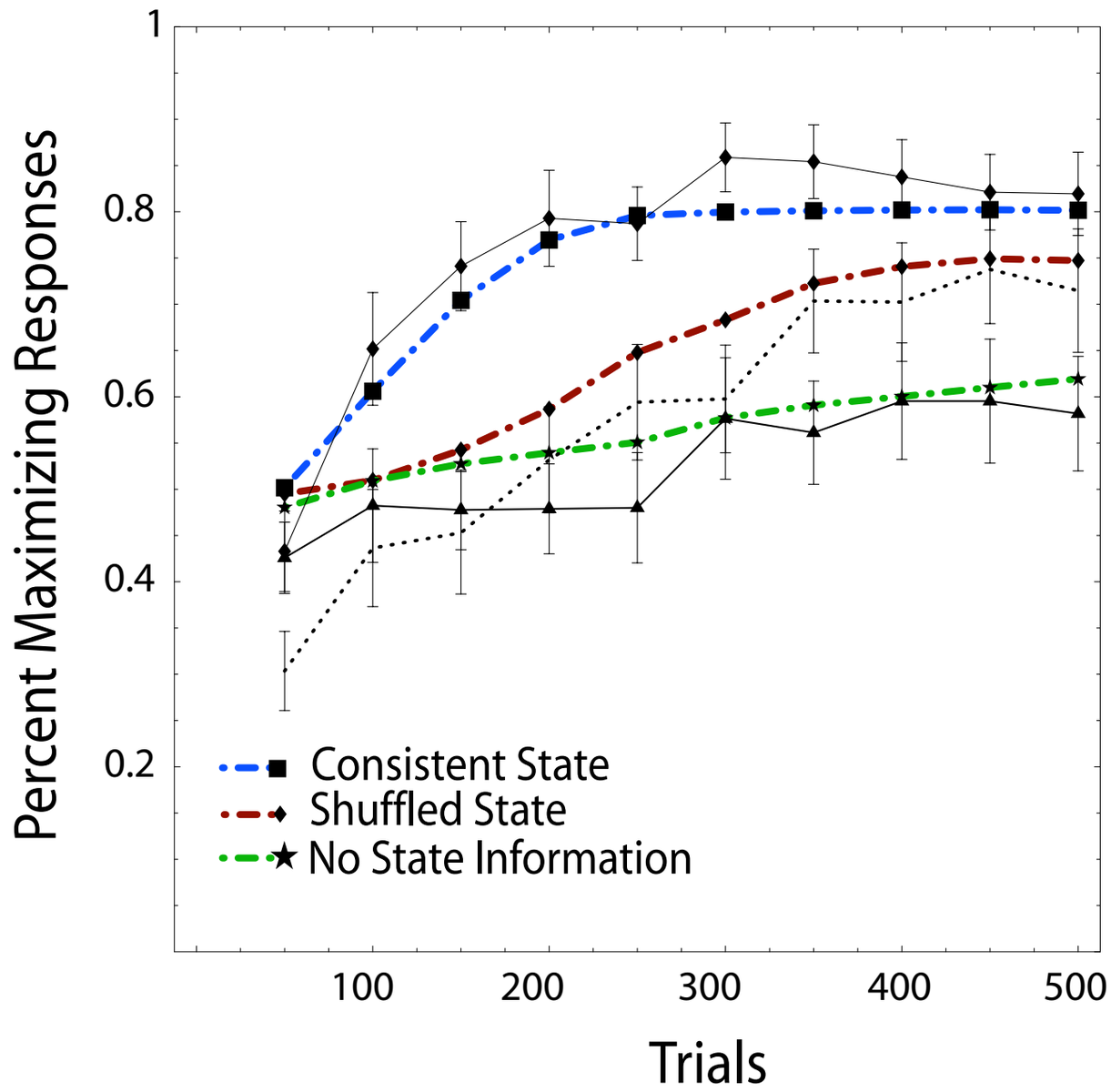

Reward
2
1.5
1
0.5
0
0.2
0.4
0.6
0.8
1
MEL
MAX
Percentage of Maximizing Respones over Last 10 Trials


WELCOME TO THE N.A.S.A. MARS FARMING PROJECT


SYSTEM 1
SYSTEM 2
Choose


SYSTEM 1
SYSTEM 2
VENT
VENT



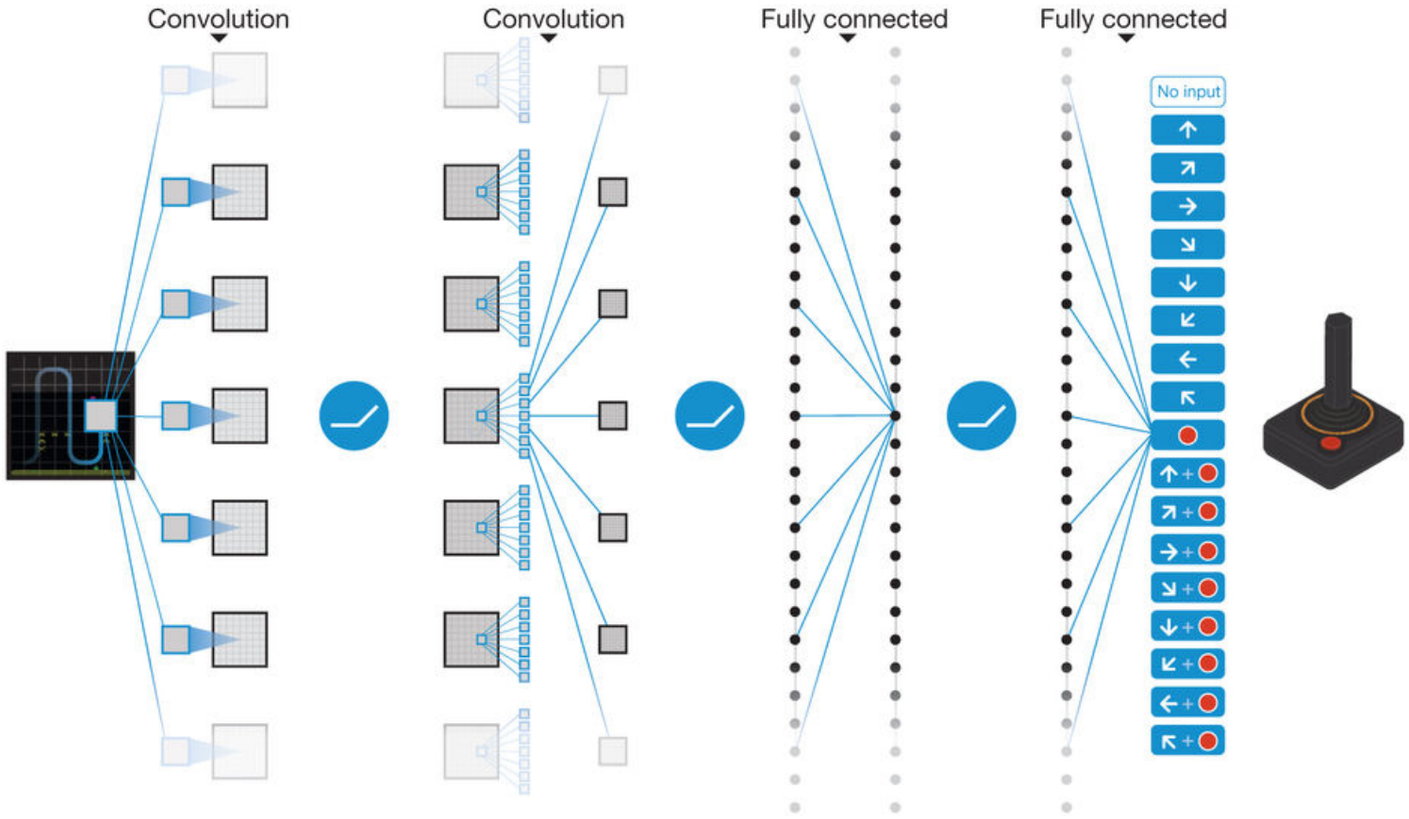


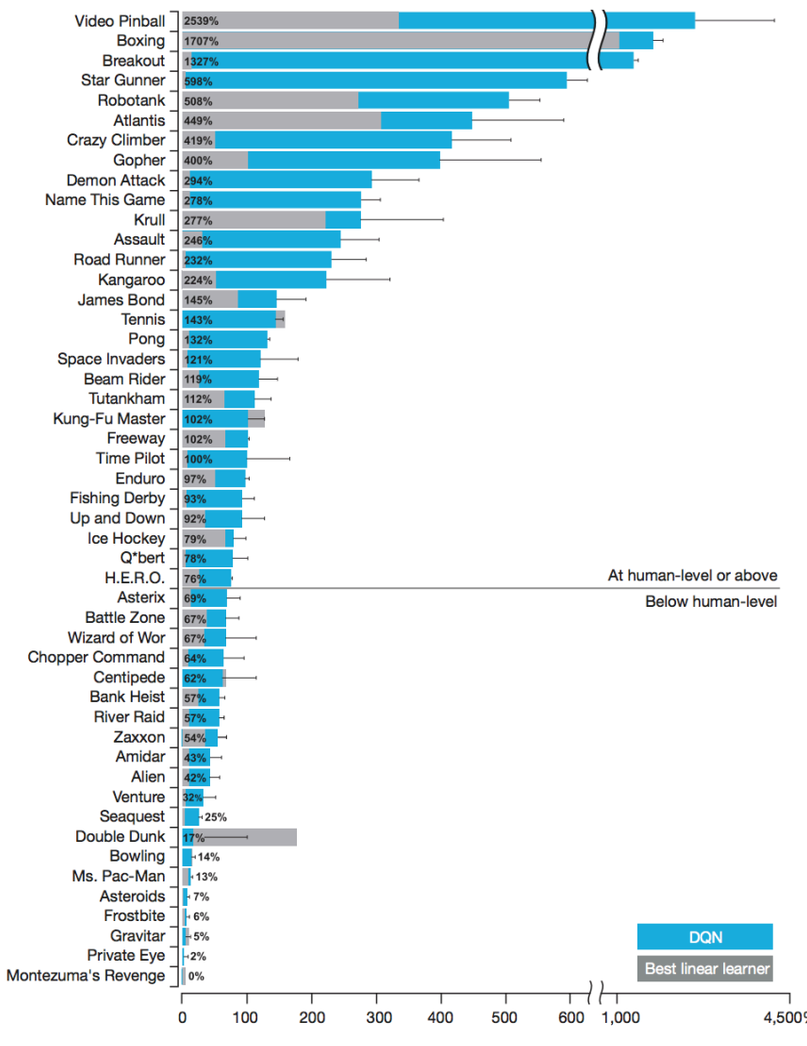
$$Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

$$L = \frac{1}{2}[r + max_{a'} Q(s', a') - Q(s, a)]^2$$

The way the problem is represented to the network makes it easier to learn

the state representation matters!





LETTER

doi:10.1038/nature14236

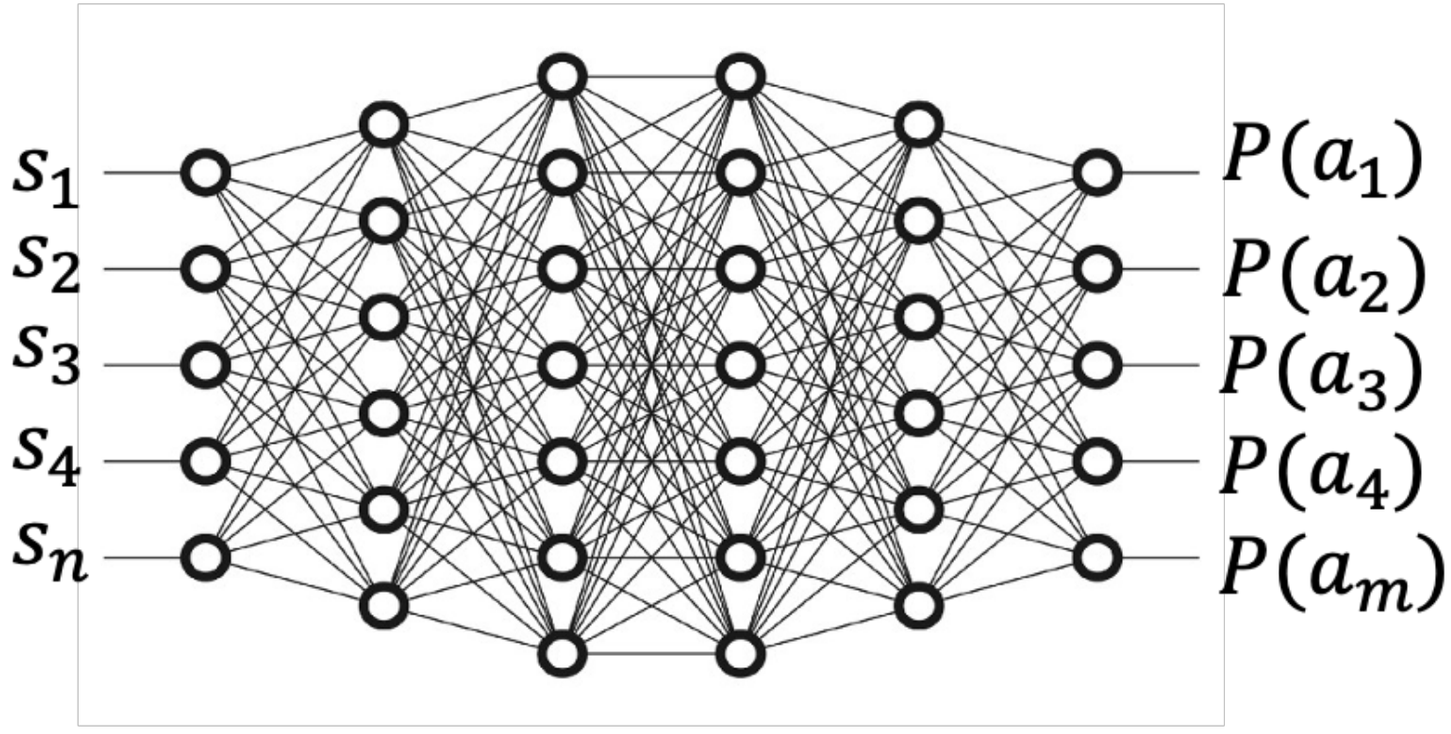
Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

DQN!

Learning Policies Directly Using Policy Gradient

- So far we have focused on learning values (v-values or Q-values) with the policy treated separately on top (e.g., epsilon-greedy or software)
- Instead what if we **learn policies directly?**
- Represent the policy as a continuous, differentiable function (e.g., an artificial neural network)



Learning Policies Directly Using Policy Gradient

- So far we have focused on learning values (v-values or Q-values) with the policy treated separately on top (e.g., epsilon-greedy or software)
- Instead what if we **learn policies directly?**
- Represent the policy as a continuous, differentiable function (e.g., an artificial neural network)

$$R_t(\tau) = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$ The return

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$ The objective

$$\max_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$ Aka, the problem to optimize

Learning Policies Directly Using Policy Gradient

- So far we have focused on learning values (v-values or Q-values) with the policy treated separately on top (e.g., epsilon-greedy or software)
- Instead what if we **learn policies directly?**
- Represent the policy as a continuous, differentiable function (e.g., an artificial neural network)

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$$

Gradient ascent in parameter Space to maximize the return

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} R_t(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right]$$

The policy gradient

** see several sources including Sutton and Barto book for proof of policy gradient theorem, Williams (1992) REINFORCE algorithm

Learning Policies Directly Using Policy Gradient

- So far we have focused on learning values (v-values or Q-values) with the policy treated separately on top (e.g., epsilon-greedy or software)
- Instead what if we **learn policies directly?**
- Represent the policy as a continuous, differentiable function (e.g., an artificial neural network)

- If the Return is greater than zero than probability of action is increased, if less than zero probability of action is decreased (Thorndike law of effect!)
- Can alter discrete action probabilities but also apply to continuous action parameters like the mean and standard deviation of a Gaussian!
- Combined neural networks with the problem of dynamic control!

Algorithm 2.1 REINFORCE algorithm

Initialize learning rate α
Initialize weights θ of a policy network π_θ
for *episode* $= 0, \ldots,$ *MAX_EPISODE* **do**
    Sample a trajectory $\tau = s_0, a_0, r_0, \ldots, s_T, a_T, r_T$
    Set $\nabla_\theta J(\pi_\theta) = 0$
    for $t = 0, \ldots, T$ **do**
        $R_t(\tau) = \sum_{t'=t}^{T} \gamma^{t'-t} r'_t$
        $\nabla_\theta J(\pi_\theta) = \nabla_\theta J(\pi_\theta) + R_t(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$
    end for
    $\theta = \theta + \alpha \nabla_\theta J(\pi_\theta)$
end for

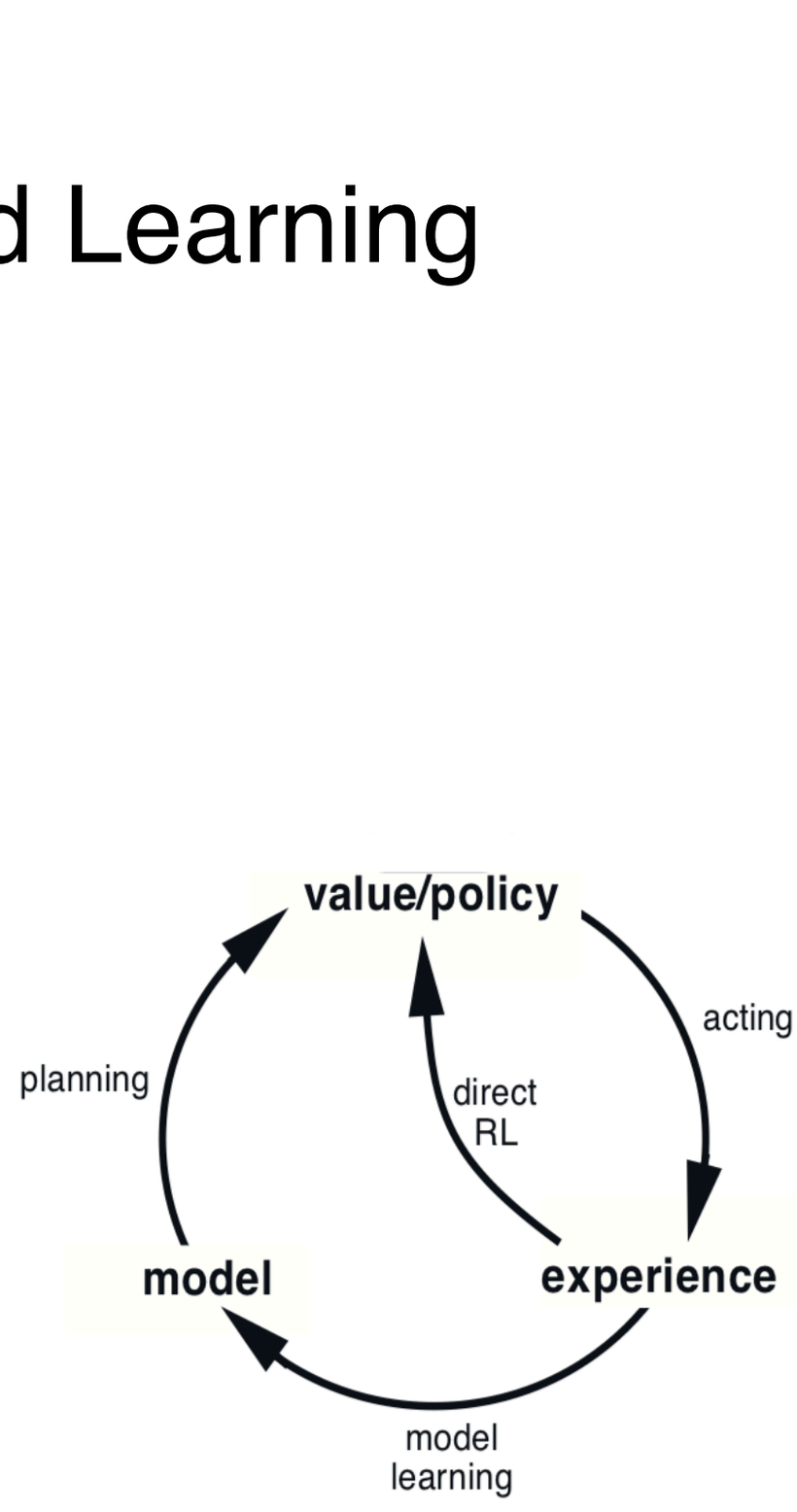
Planning or Model-based Learning

Temporal-Difference/Monte-carlo methods we have considered so far are call “model-free” because they don’t maintain or learn information about the state transitions and rewards explicitly.

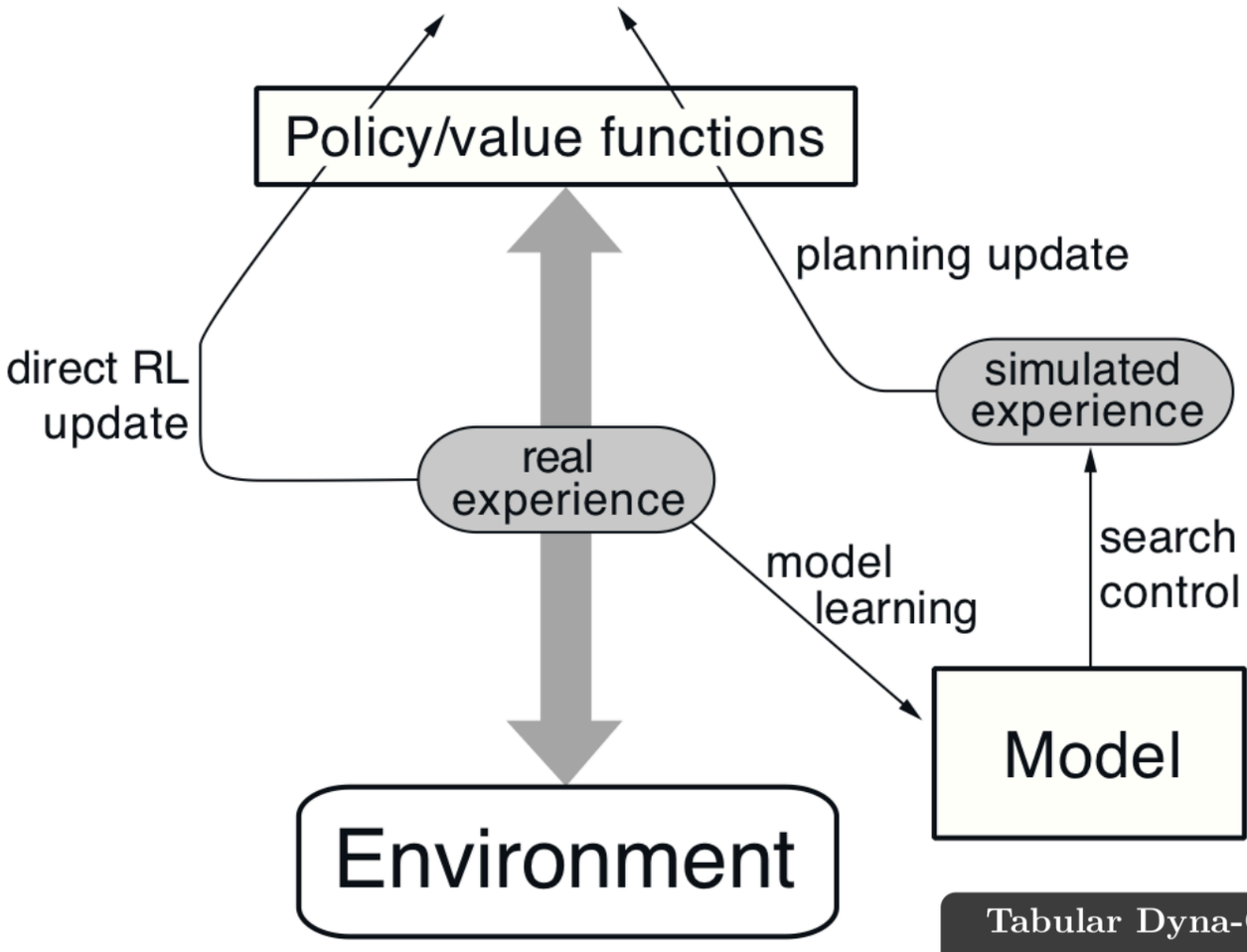
In contrast, Dynamic Programming needs a model of the environment in terms of the transition probabilities and rewards.

If you have a model you can distinguish **planning** from **learning.**

Planning - predict what will happen in the future… what state will I end up in, what reward will I get?



The DYNA algorithm



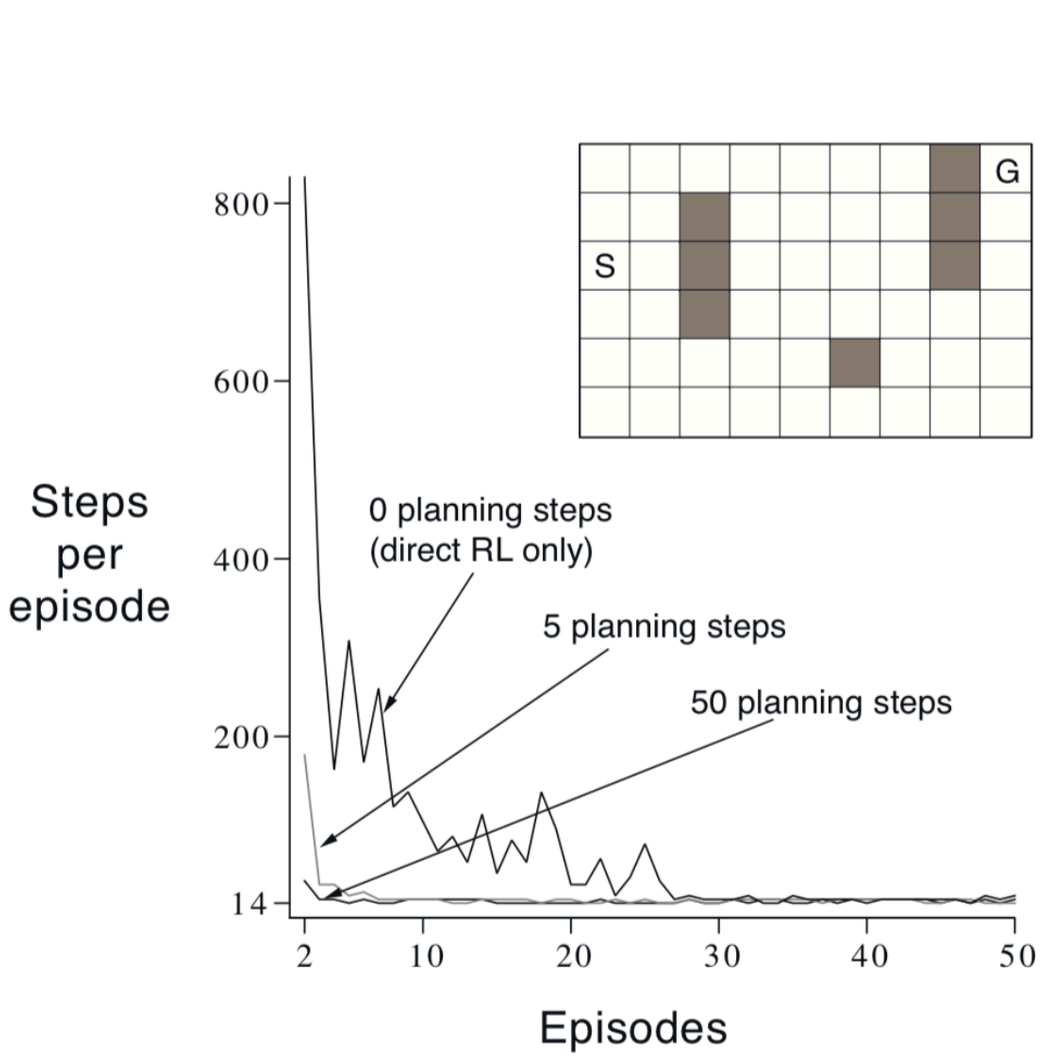
Combines direct experience which helps learn direct RL value/policy updates with model learning.

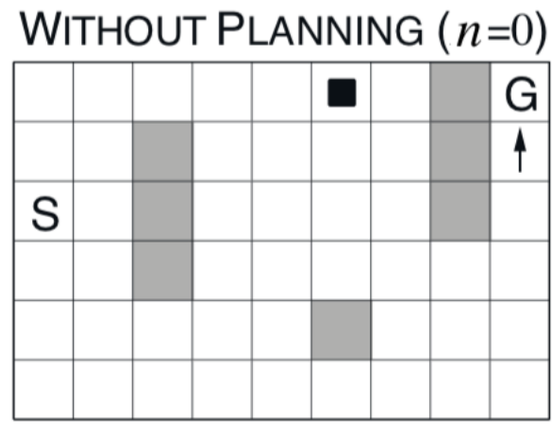
Simulated experience used to aid planning.

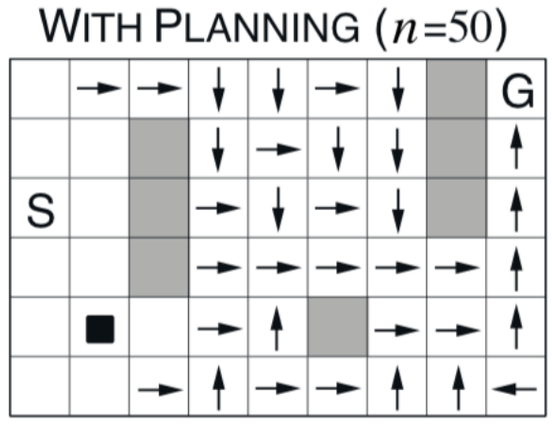
Tabular Dyna-Q

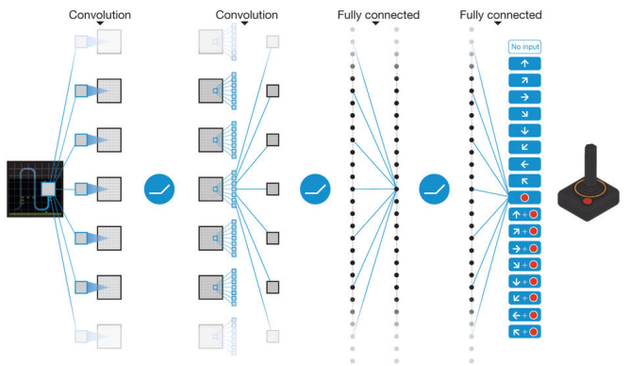
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \varepsilon$-greedy(S,Q)
- (c) Take action A; observe resultant reward, R, and state, S'
- (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
- (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
- (f) Loop repeat n times:
  - $S \leftarrow$ random previously observed state
  - $A \leftarrow$ random action previously taken in S
  - $R, S' \leftarrow Model(S,A)$
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

The DYNA algorithm



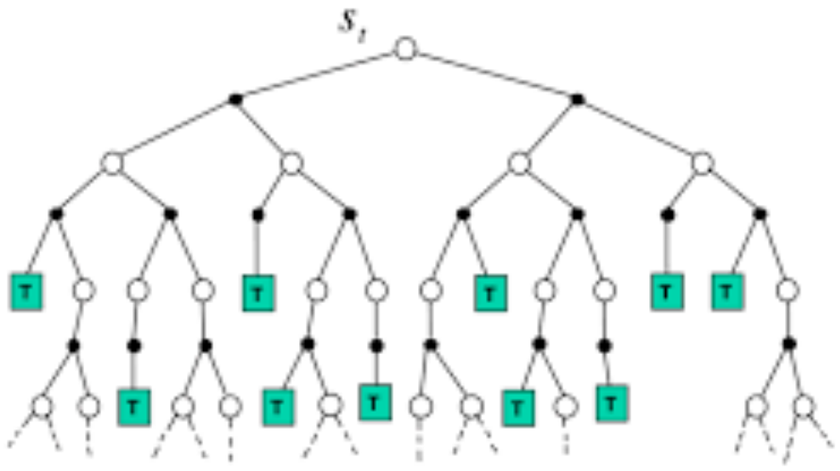






"experience replay" also used in Atari Solution

Online planning



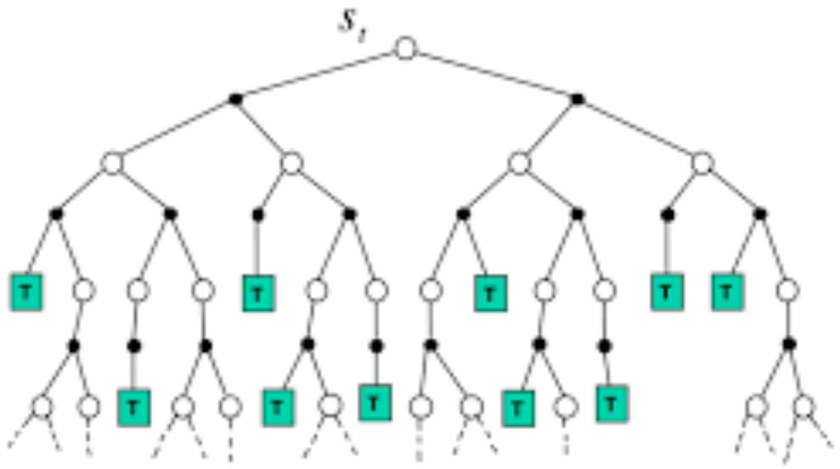
From each state, if we have a good model of environment we can simulate forward many times to determine the value of the current state or action.

However, the branching factor of this tree can be a major problem.

One recent innovation is **Monte Carlo Tree Search (MCTS)**

jeffbradberry.com (Hacker Monthly)

Monte Carlo Tree Search



$$\bar{x}_i \pm \sqrt{\frac{2 \ln n}{n_i}}$$

- $\bar{x}_i$: the mean payout for machine i
- n_i: the number of plays of machine i
- n: the total number of plays

- Pick option with highest confidence bound

UCB1

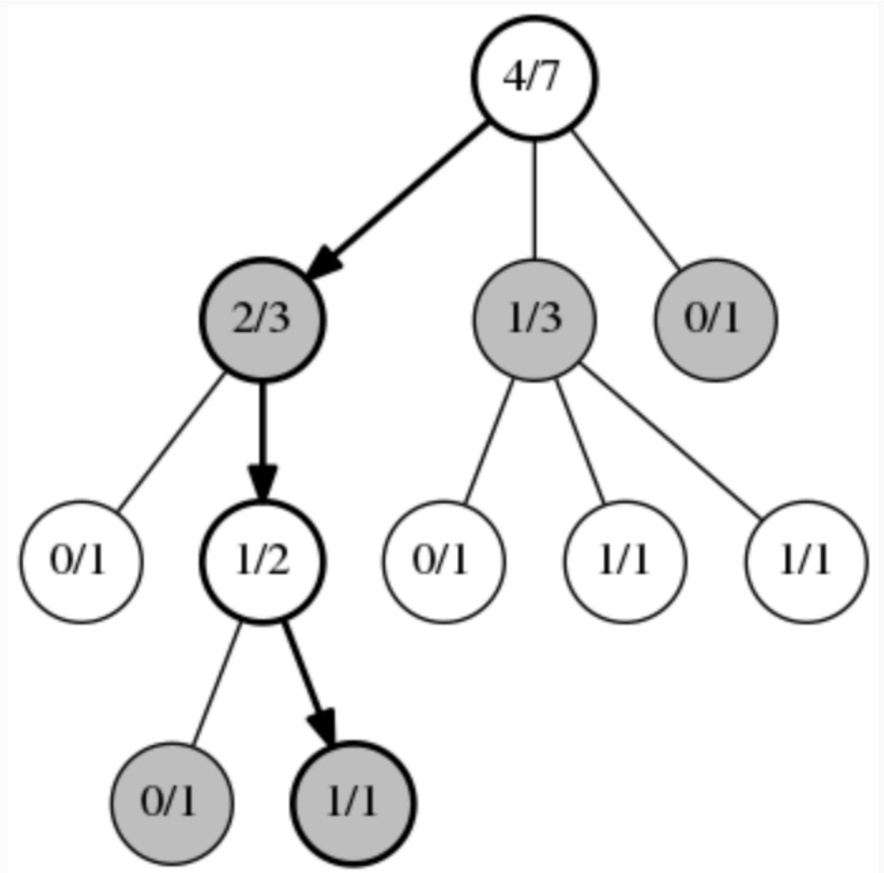
- In a standard Monte Carlo process, a large number of random simulations are run, in this case, from the board position that you want to find the best move for.

- The downside to this method, though, is that for any given turn in the simulation, there may be many possible moves, but only one or two that are good.

- Instead of doing many purely random simulations, UCB works by doing many multi-phase playouts where each position is itself viewed as a multi-armed bandit.

jeffbradberry.com (Hacker Monthly)

Monte Carlo Tree Search



Selection

Here the positions and moves selected by the UCB1 algorithm at each step are marked in bold. Note that a number of playouts have already been run to accumulate the statistics shown. Each circle contains the number of wins / number of times played.

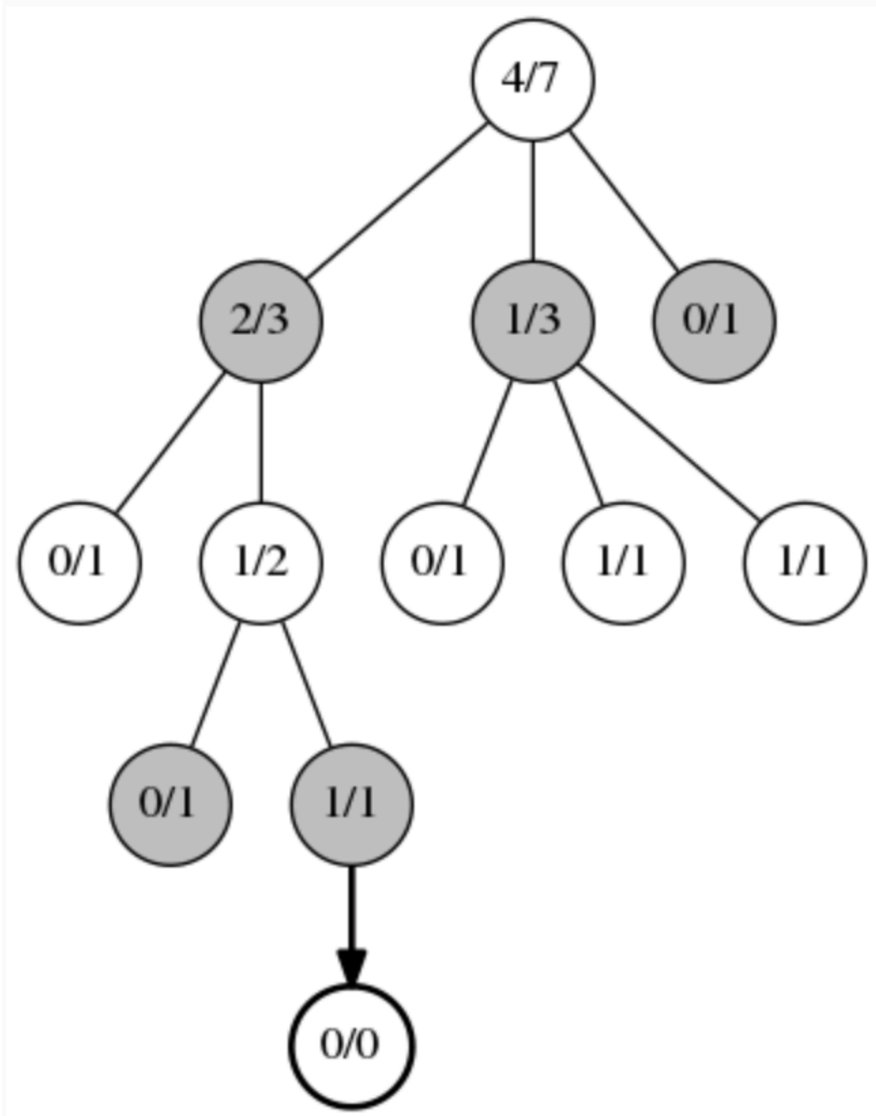
$$\bar{x}_i \pm \sqrt{\frac{2 \ln n}{n_i}}$$

- $\bar{x}_i$: the mean payout for machine i
- n_i: the number of plays of machine i
- n: the total number of plays

UCB1

jeffbradberry.com (Hacker Monthly)

Monte Carlo Tree Search



Expansion

The position marked 1/1 at the bottom of the tree has no further statistics records under it, so we choose a random move and add a new record for it (bold), initialized to 0/0.

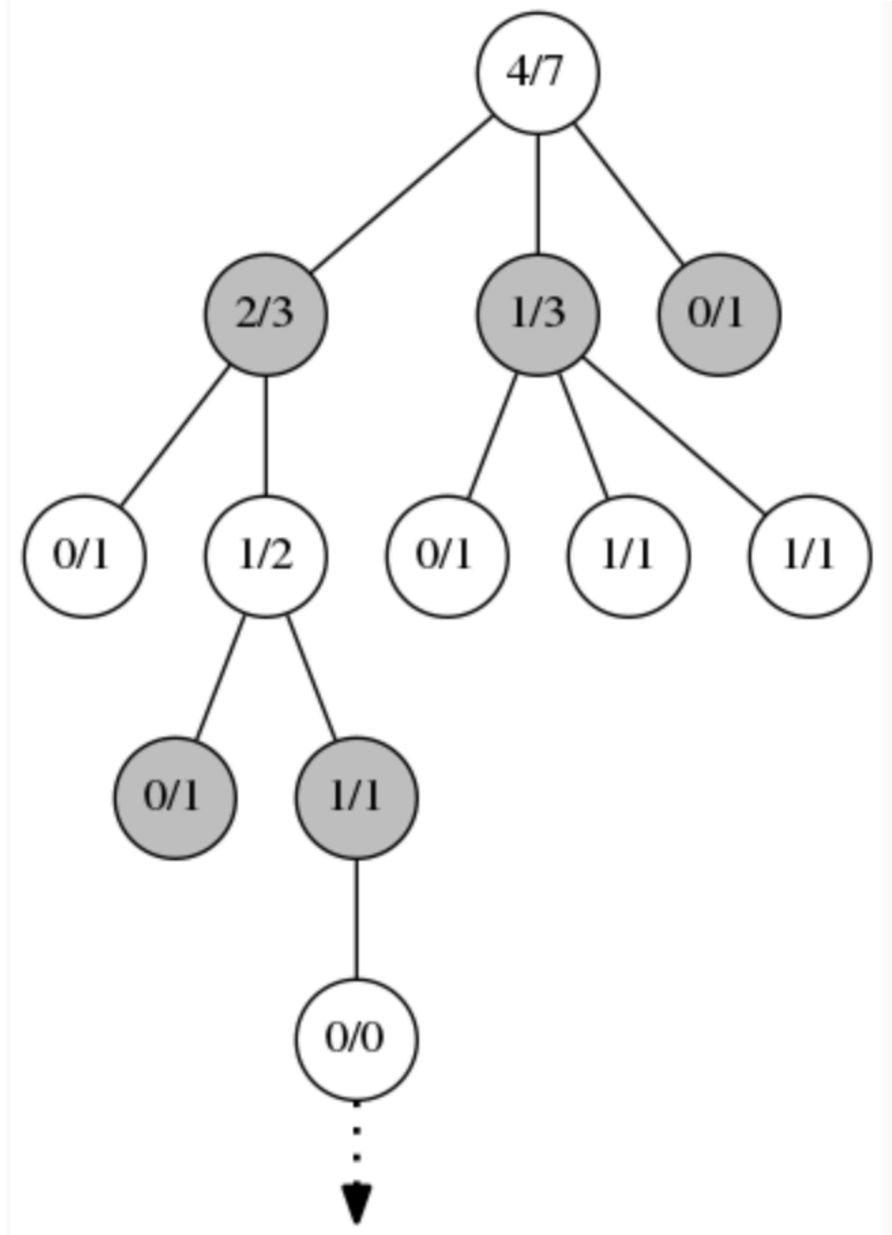
$$\bar{x}_i \pm \sqrt{\frac{2 \ln n}{n_i}}$$

- $\bar{x}_i$: the mean payout for machine i
- n_i: the number of plays of machine i
- n: the total number of plays

UCB1

jeffbradberry.com (Hacker Monthly)

Monte Carlo Tree Search



Simulation

Once the new record is added, the Monte Carlo simulation begins, here depicted with a dashed arrow. Moves in the simulation may be completely random, or may use calculations to weight the randomness in favor of moves that may be better.

$$\bar{x}_i \pm \sqrt{\frac{2 \ln n}{n_i}}$$

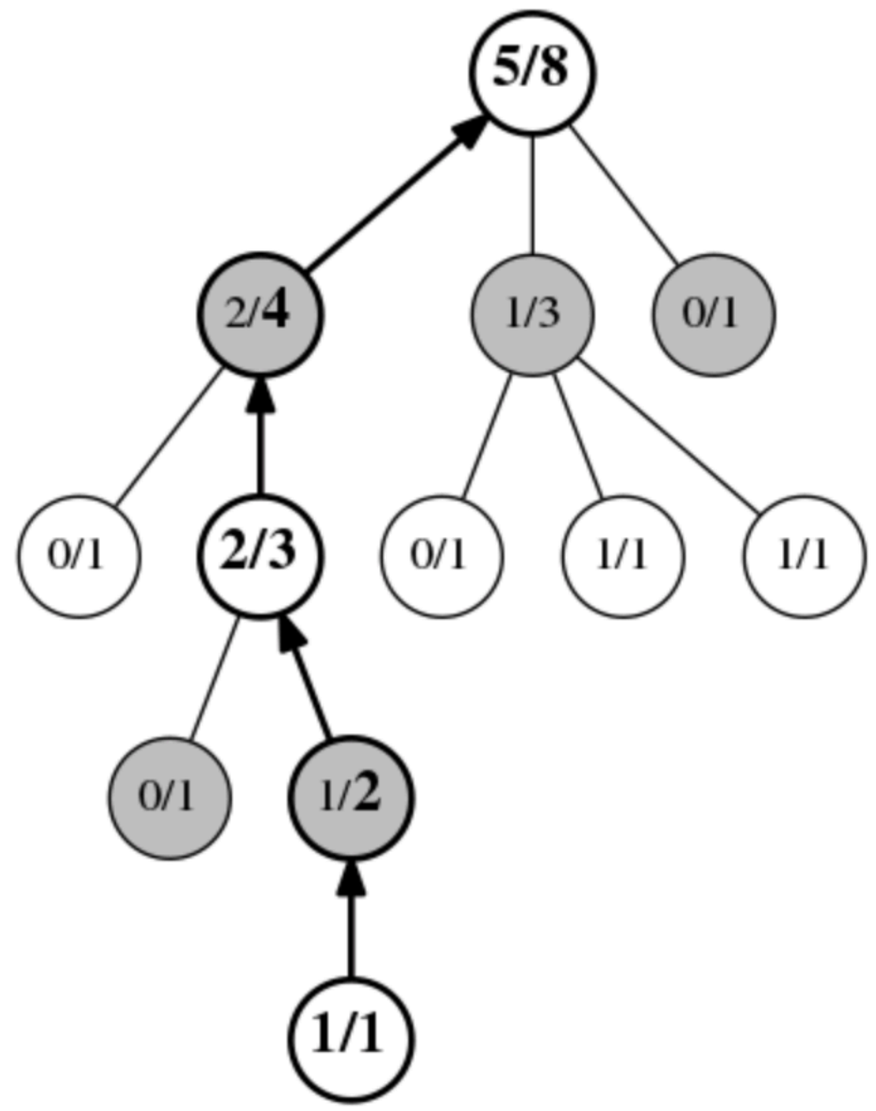
- $\bar{x}_i$: the mean payout for machine i
- n_i: the number of plays of machine i
- n: the total number of plays

UCB1

jeffbradberry.com (Hacker Monthly)

Monte Carlo Tree Search

- As more and more playouts are run, the tree of statistics grows in memory and the move that will finally be chosen will converge towards the actual optimal play, though that may take a very long time, depending on the game.



Back-Propagation

After the simulation reaches an end, all of the records in the path taken are updated. Each has its play count incremented by one, and each that matches the winner has its win count incremented by one, here shown by the bolded numbers.

$$\bar{x}_i \pm \sqrt{\frac{2 \ln n}{n_i}}$$

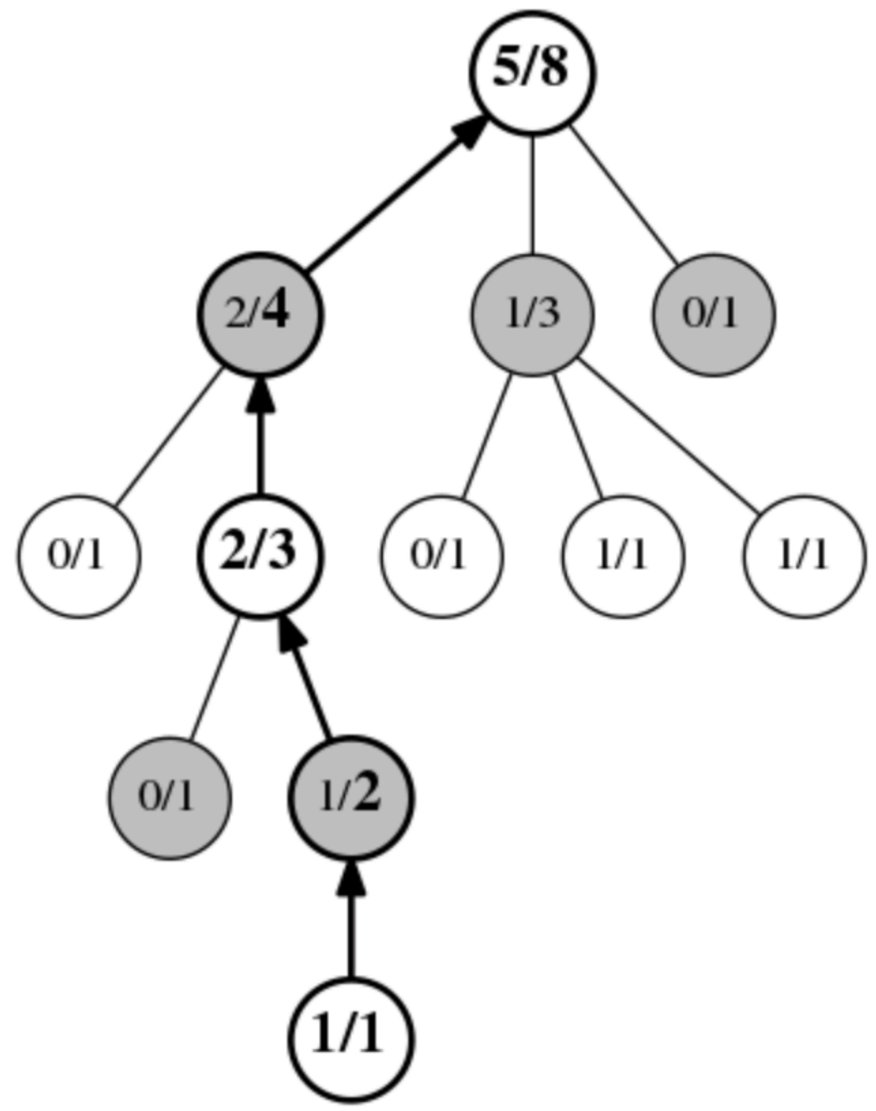
- $\bar{x}_i$: the mean payout for machine i
- n_i: the number of plays of machine i
- n: the total number of plays

UCB1

jeffbradberry.com (Hacker Monthly)

Monte Carlo Tree Search

- As more and more playouts are run, the tree of statistics grows in memory and the move that will finally be chosen will converge towards the actual optimal play, though that may take a very long time, depending on the game.



Back-Propagation

After the simulation reaches an end, all of the records in the path taken are updated. Each has its play count incremented by one, and each that matches the winner has its win count incremented by one, here shown by the bolded numbers.

$$\bar{x}_i \pm \sqrt{\frac{2 \ln n}{n_i}}$$

- $\bar{x}_i$: the mean payout for machine i
- n_i: the number of plays of machine i
- n: the total number of plays

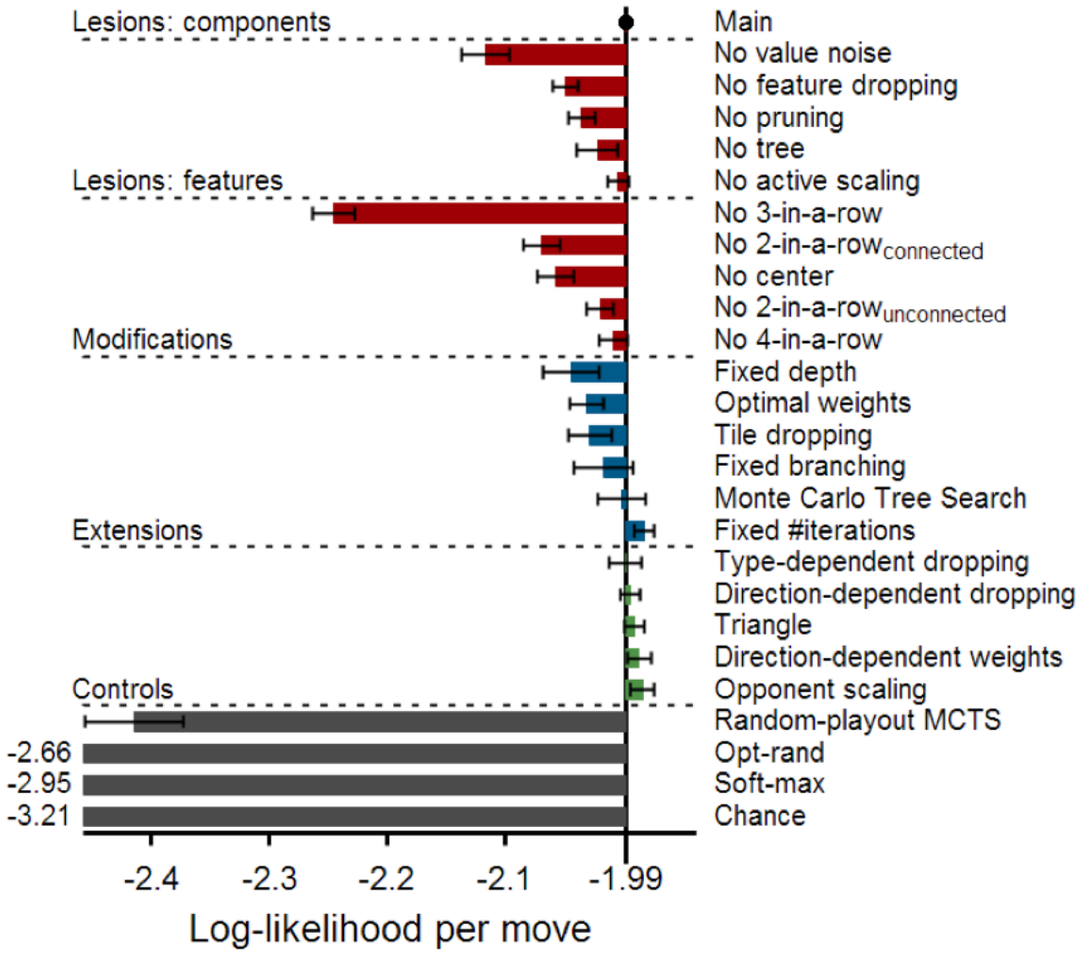
UCB1

jeffbradberry.com (Hacker Monthly)

Human Tree Search?

Van Opheusden B, Galbiati G, Bnaya Z, Li Y, Ma WJ (2017),
A computational model for decision tree search.
Proceedings of the 39th Annual Meeting of the Cognitive Science Society, 1254-1259.

$$V(s) = c_{\text{self}} \sum_{i=0}^{4} w_i f_i(s, \text{self}) - c_{\text{opp}} \sum_{i=0}^{4} w_i f_i(s, \text{opponent})$$



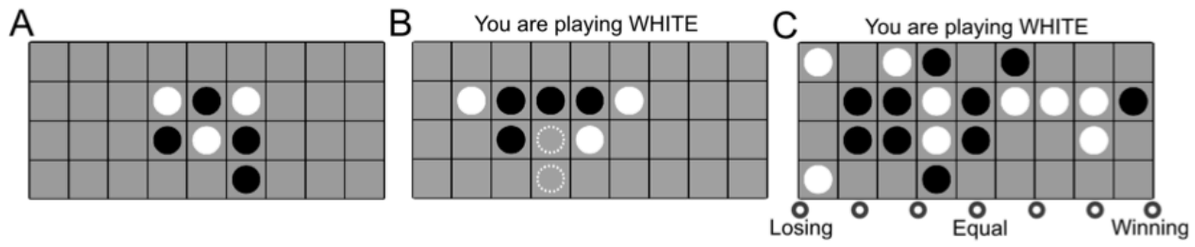
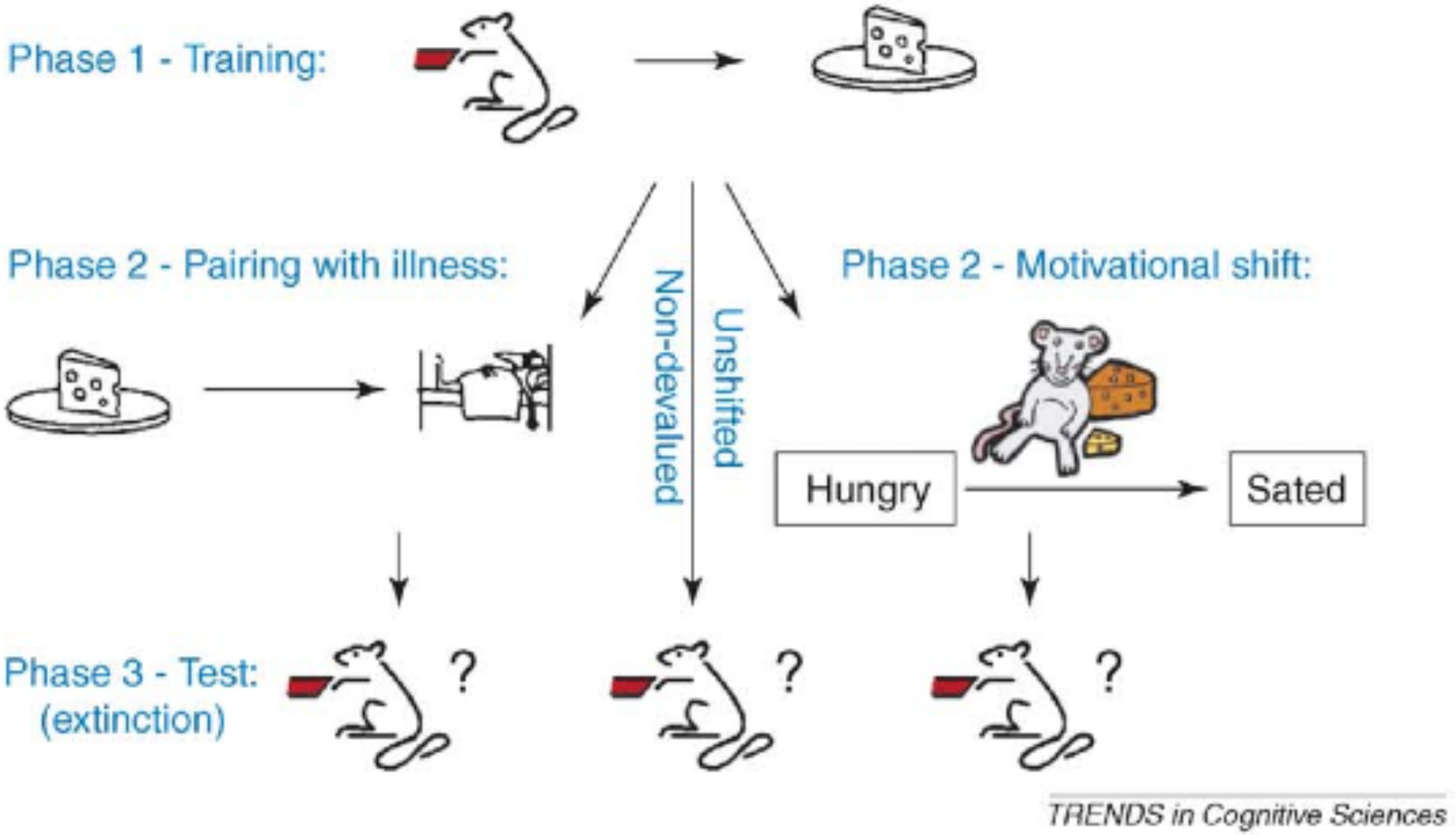


Figure 1: Task. **A.** Two players take turns placing black or white pieces on a 4-by-9 board, and the first to achieve 4-in-a-row (horizontal, diagonal or vertical), wins the game. **B.** In the 2AFC task, participants see a board and two candidate moves, and indicate their preferences. **C.** In the evaluation task, participants see a board position and report their estimated winning chances on a 7-point scale.

https://www.cns.nyu.edu/malab/static/files/publications/2017%20Van%20Opheusden%20Galbiati%20Bnaya%20Li%20Ma.pdf

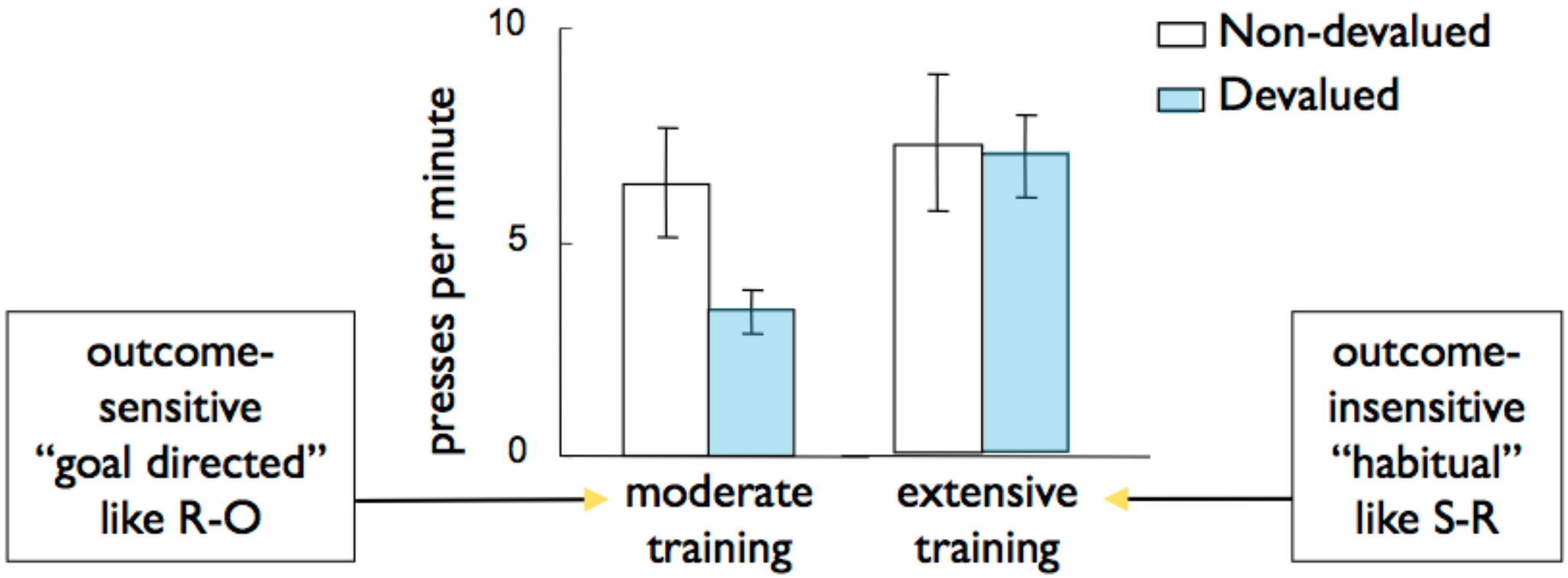
Do animals plan?



TRENDS in Cognitive Sciences

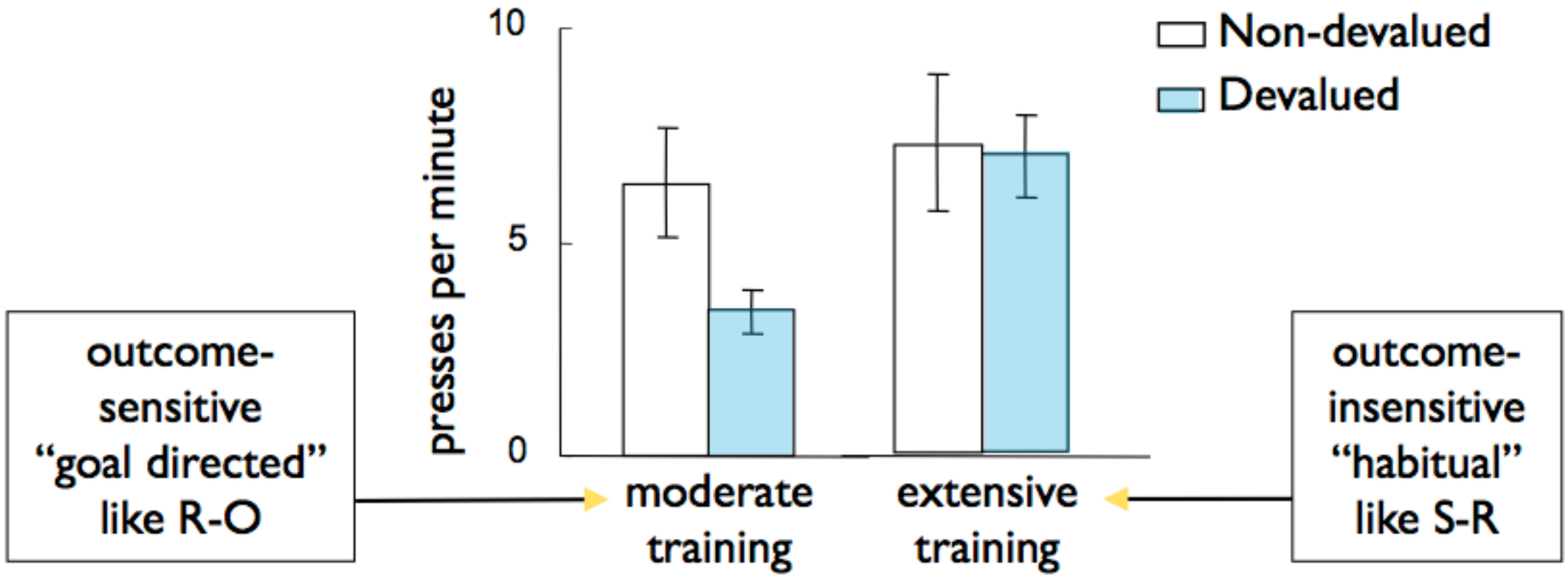
A good test: Outcome devaluation

Holland (2004) see also Dickinson article for examples



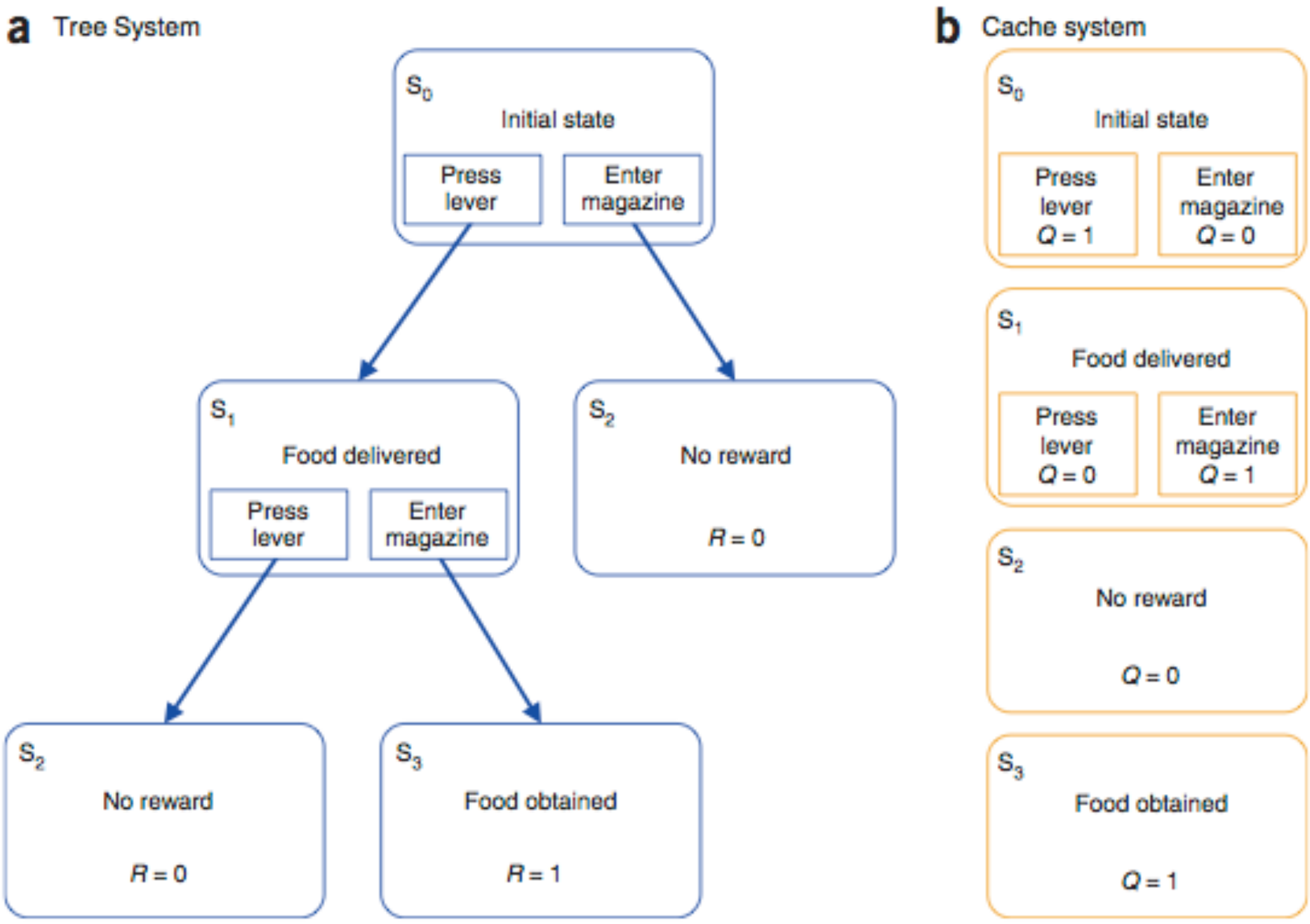
A good test: Outcome devaluation

Holland (2004) see also Dickinson article for examples



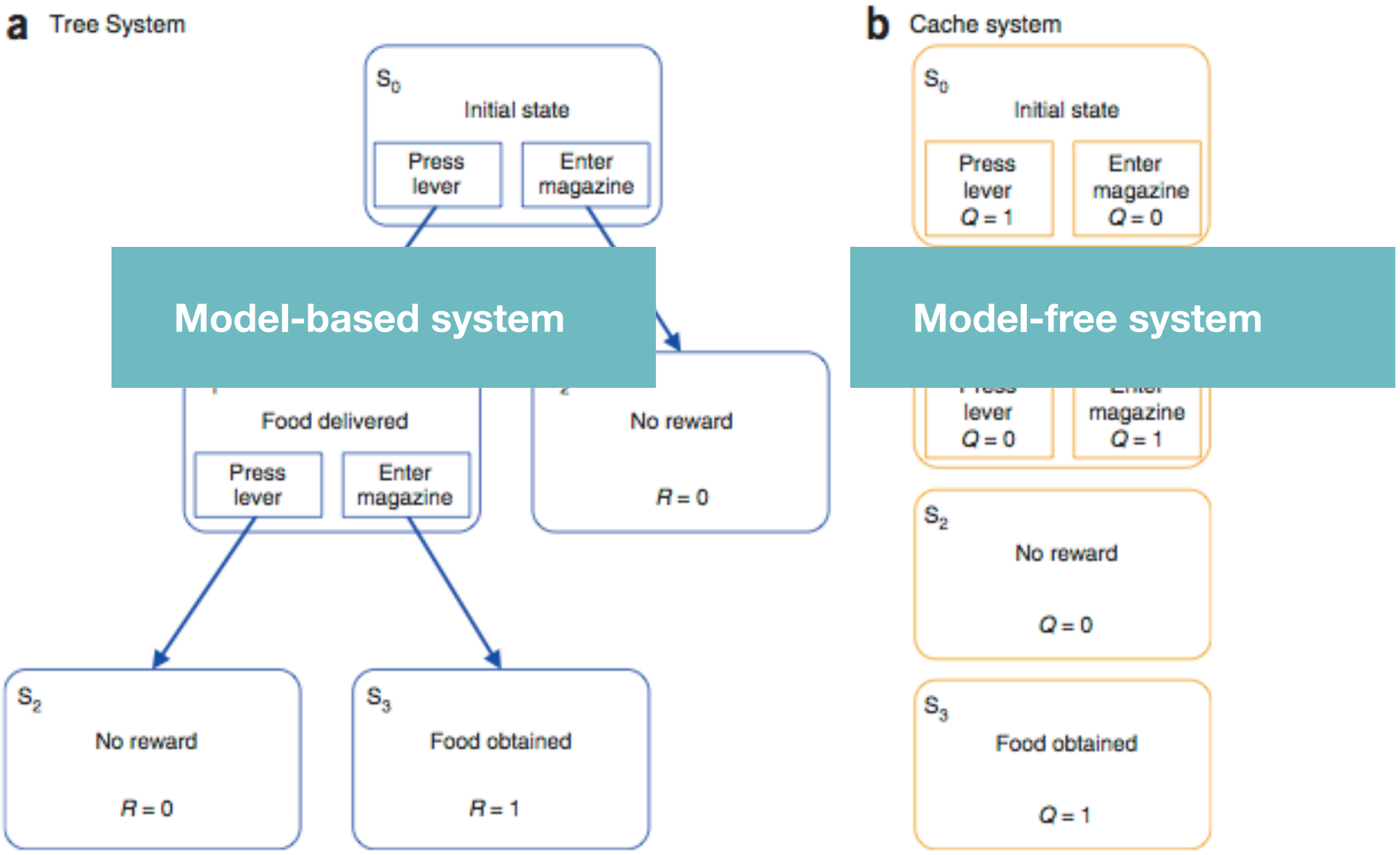
Suggests some measure of reward or outcome expectancy (shifts to habitual control with extensive training)

Key idea



from Daw, Niv, Dayan, 2005

Key idea



from Daw, Niv, Dayan, 2005

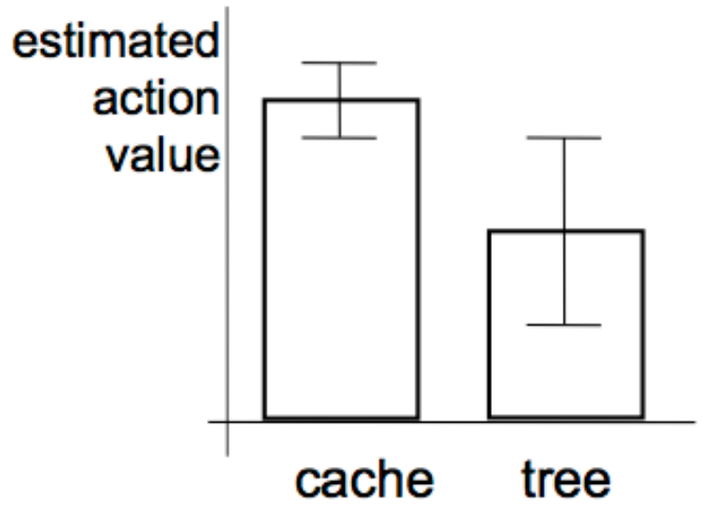
two big questions

- Why should the brain use two different strategies/ controllers in parallel?
- If it uses two: how can it arbitrate between the two when they disagree *(new decision making problem…)*

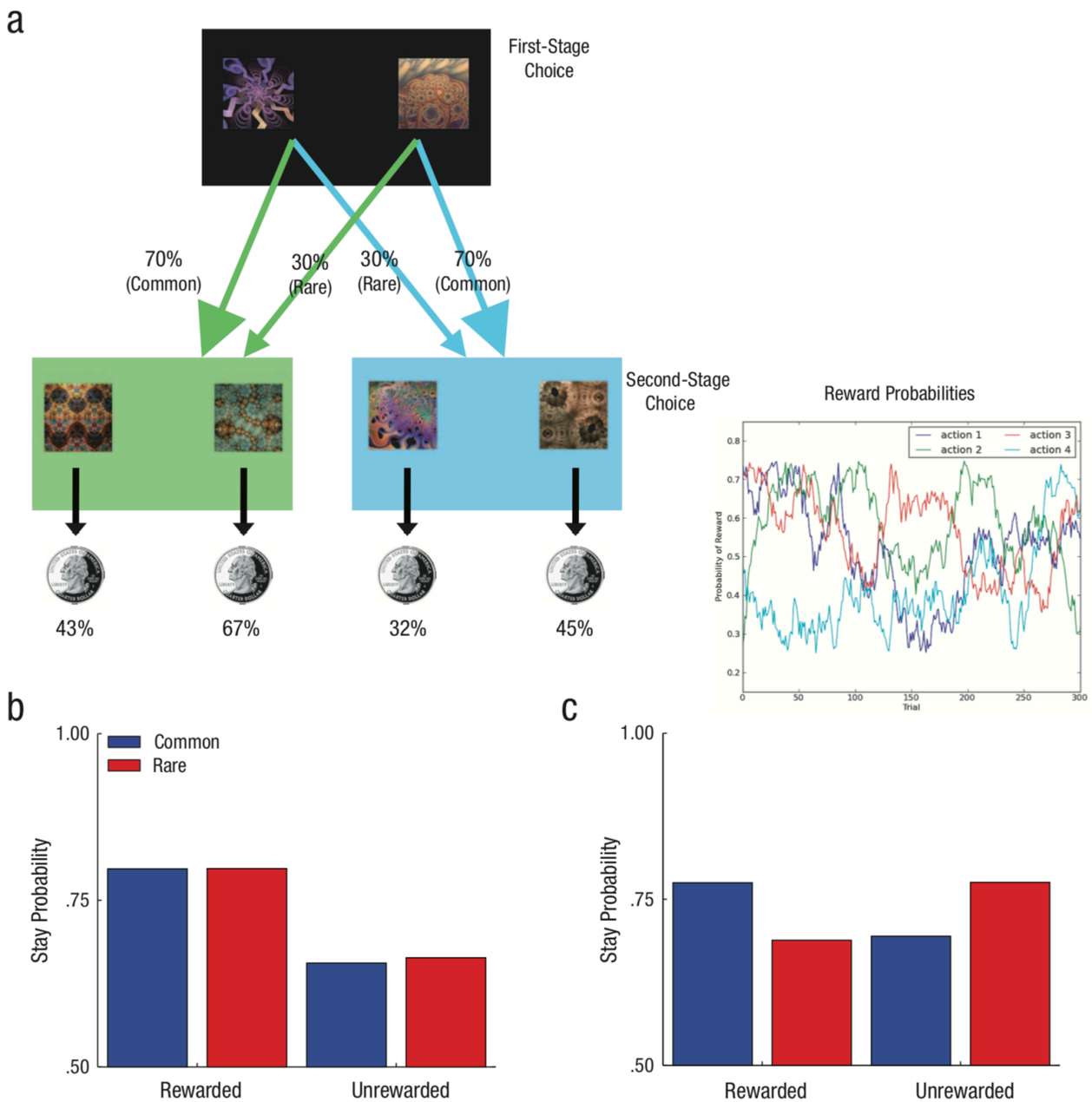
from Yael Niv

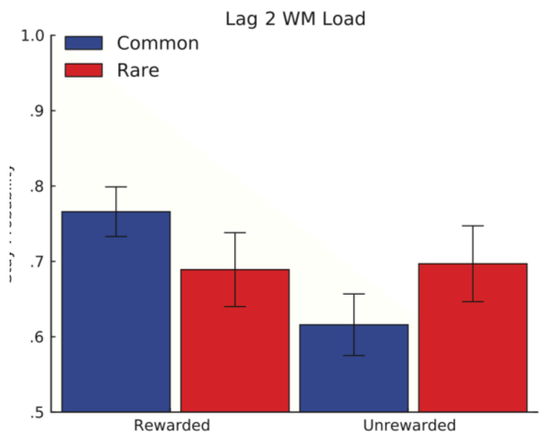
answers

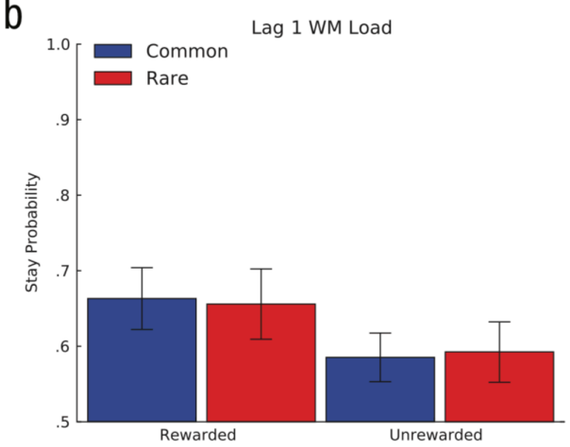
- each system is best in different situations (use each one when it is most suitable/most accurate)
  - goal-directed (forward search) - good with limited training, close to the reward (don't have to search ahead too far)
  - habitual (cache) - good after much experience, distance from reward not so important
- arbitration: trust the system that is more confident in its recommendation
  - different sources of uncertainty in the two systems
  - don't always choose the highest value
  - uncertainty is different from risk

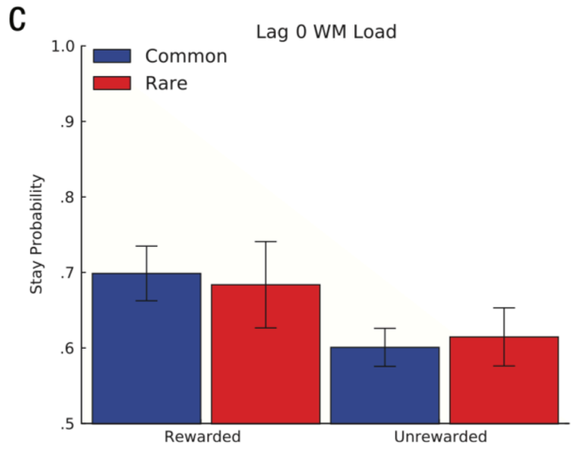


from Yael Niv









Working memory load interferes with online planning making human act more like model-free learners.

The Curse of Planning: Dissecting Multiple Reinforcement-Learning Systems by Taxing the Central Executive

A. Ross Otto[1], Samuel J. Gershman[2], Arthur B. Markman[1], and Nathaniel D. Daw[3]

[1]Department of Psychology, University of Texas at Austin; [2]Department of Psychology and Princeton Neuroscience Institute, Princeton University; and [3]Department of Psychology and Center for Neural Science, New York University

Three levels of description *(David Marr, 1982)*

Computational

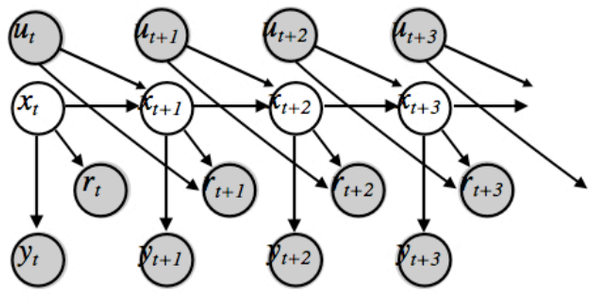
Why do things work the way they do?
What is the goal of the computation?
What are the unifying principles?

$$\textit{maximize:}\quad R_t = r_{t+1} + r_{t+2} + \cdots + r_T$$

Bellman

Algorthmic

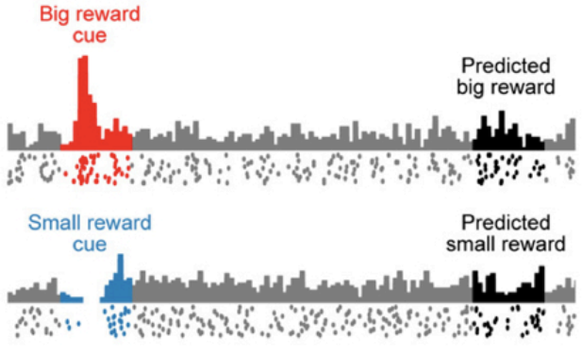
What representations can implement such computations?
How does the choice of representations determine the algorithm?



Dynamic programming, TD methods, Monte Carlo

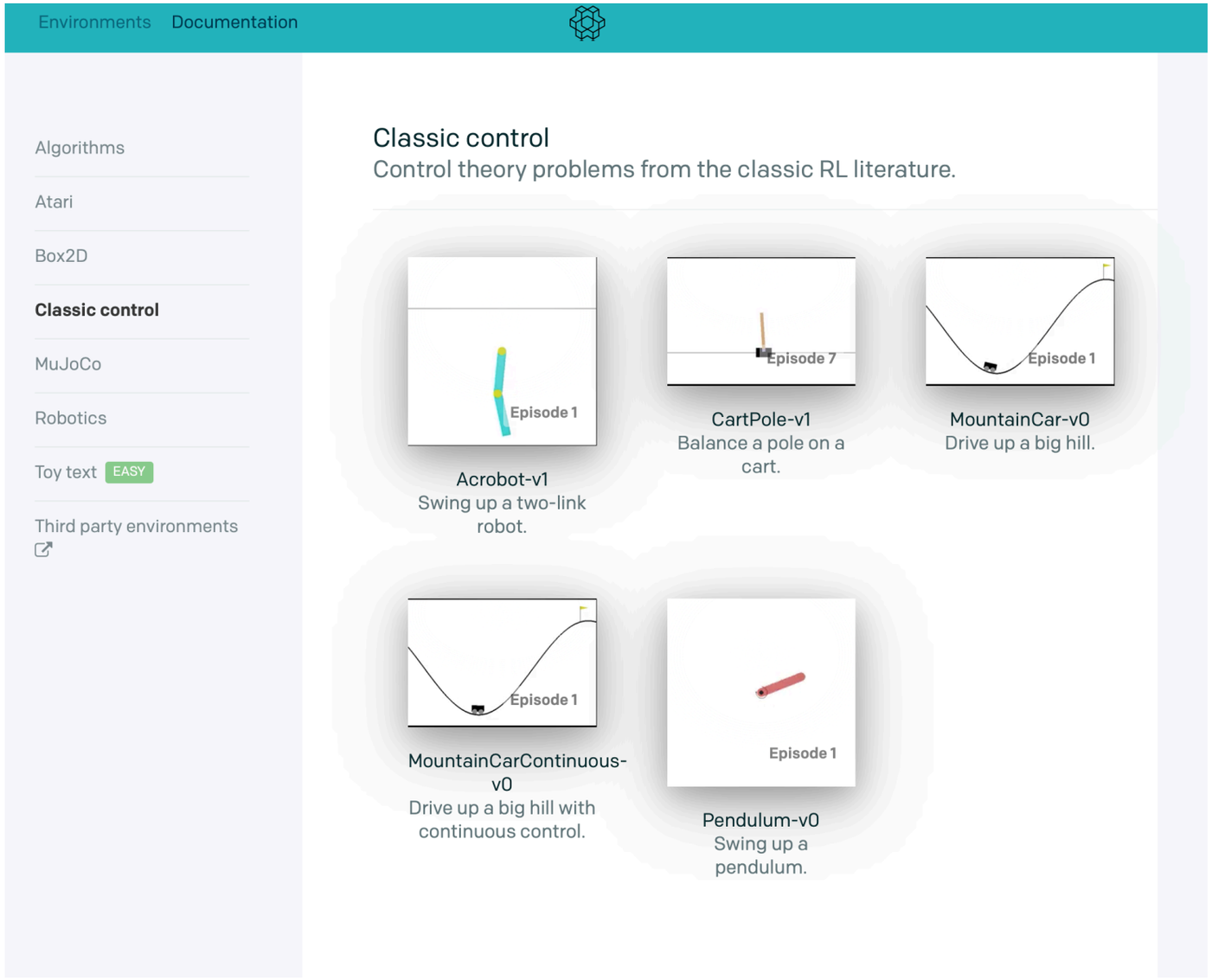
Implementational

How can such a system be built in hardware?
How can neurons carry out the computations?



Neural firing patterns, prediction errors, system level neuroscience

Playing around with RL…



Next time: Bayes

Slide Credits

Nathaniel Daw (exploration/gittins)
Alex Rich
Gillian Hayes (TD methods/explore)
Rich Sutton (general approach)
Andy Barto (general approach)