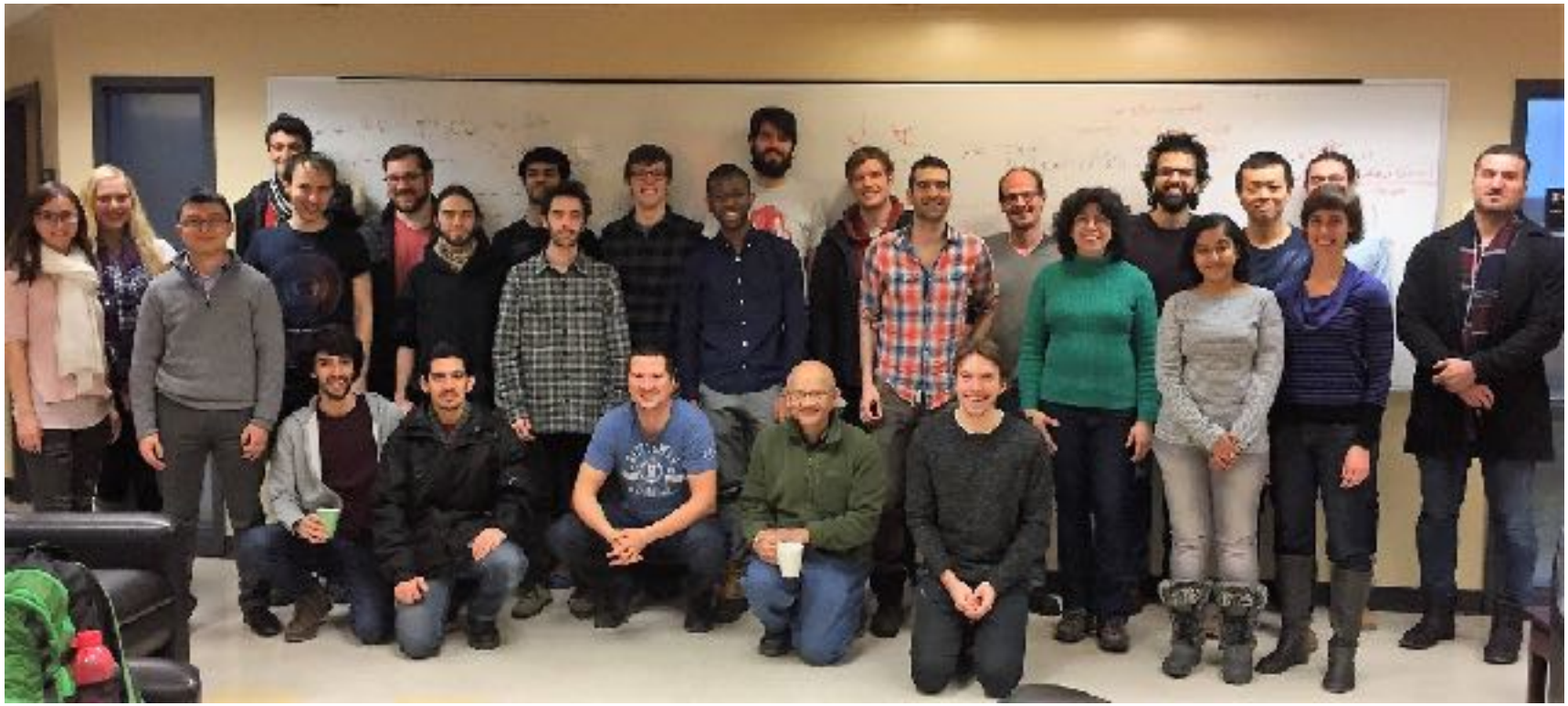
Machine learning models for temporal prediction and decision making in the brain

Doina Precup, PhD
Co-Director, Reasoning & Learning Lab, McGill University
Research Team Lead, DeepMind Montreal
Senior Member, AAAI and CIFAR; Member of MILA



McGill

Example: AlphaGo

ARTICLE

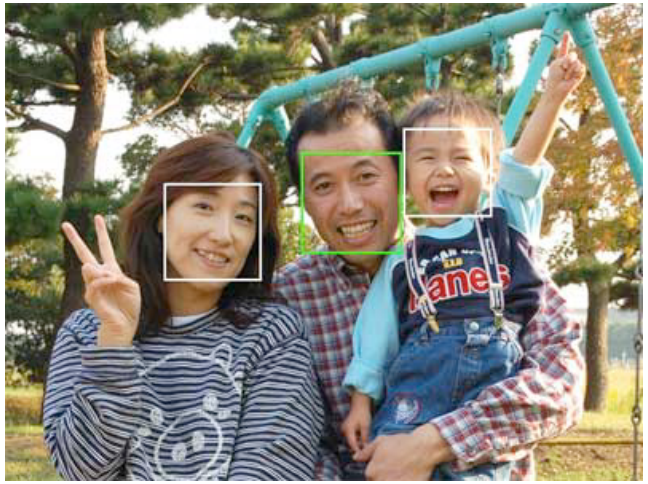
doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

McGill

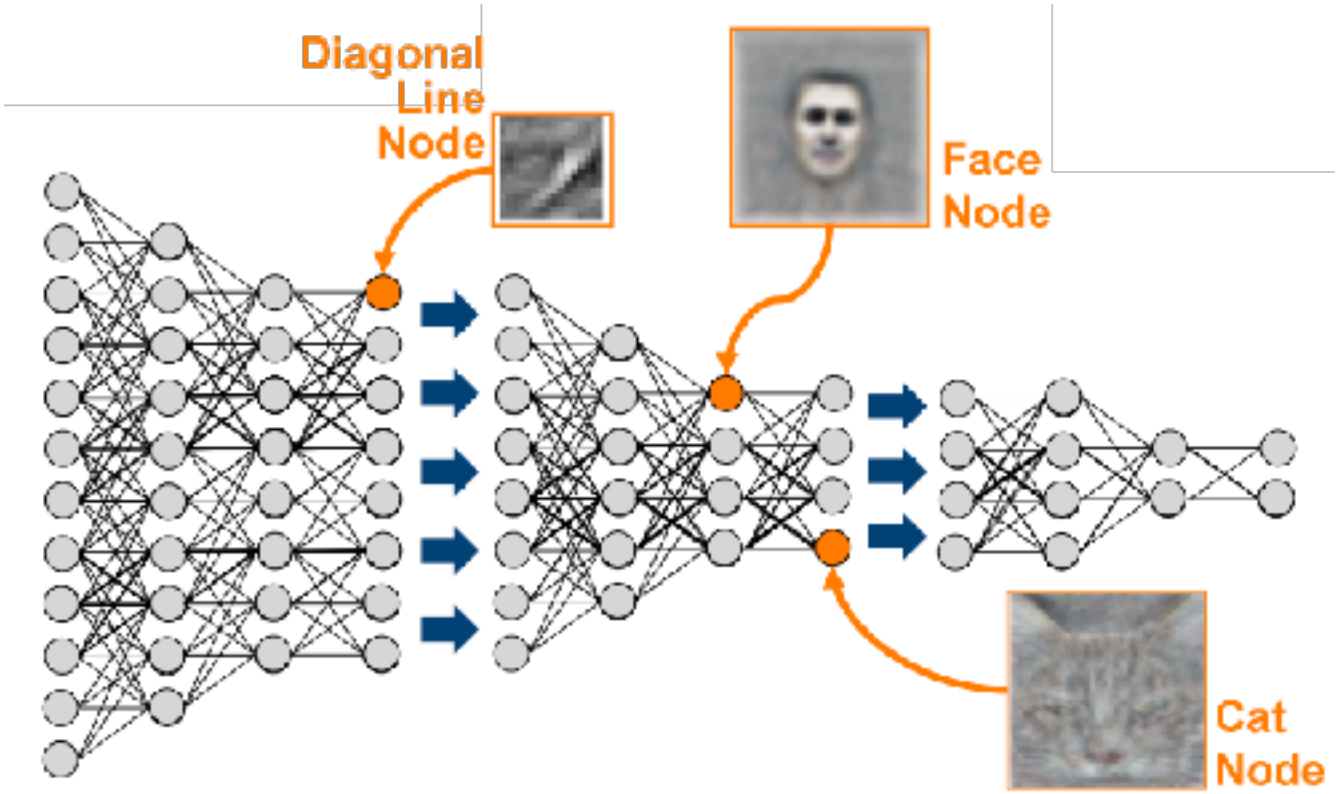
Supervised Learning



Given: Input data and desired output
Eg: Images and parts of interest

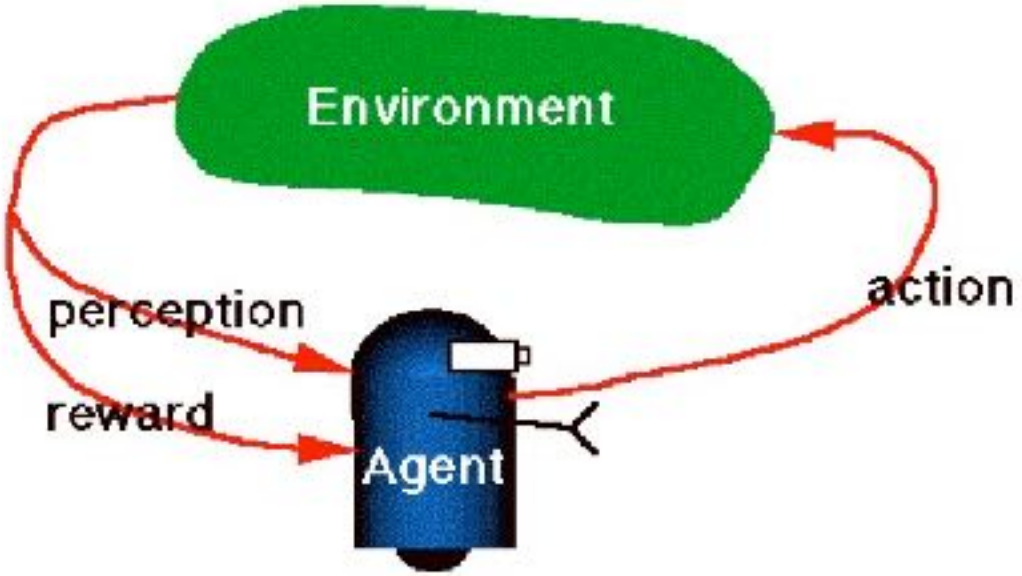
Goal: find a function that can be used for new inputs, and that matches the provided examples

Deep learning



McGill

Reinforcement Learning

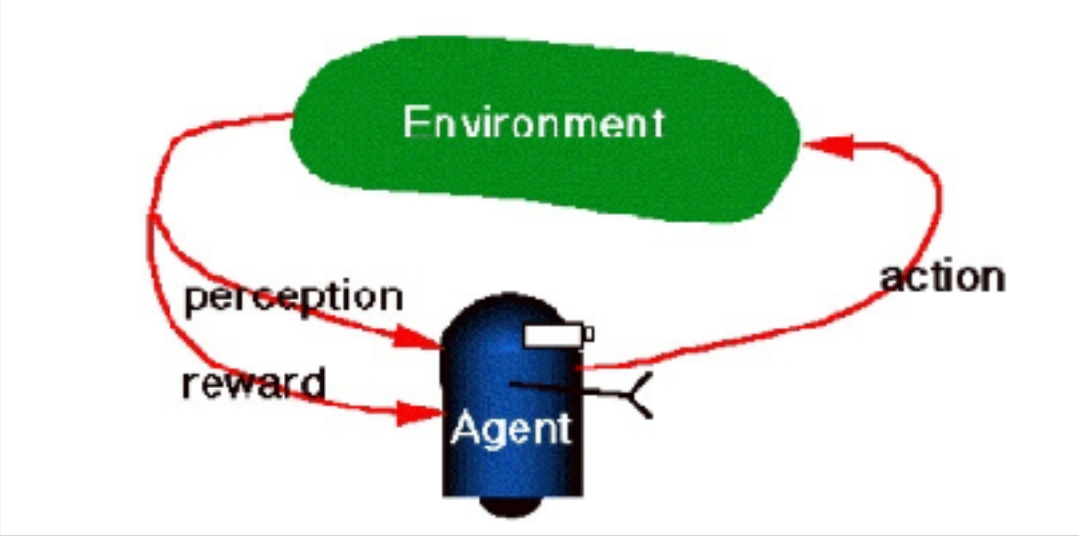


Reward: Food or shock

Reward: Positive and negative numbers

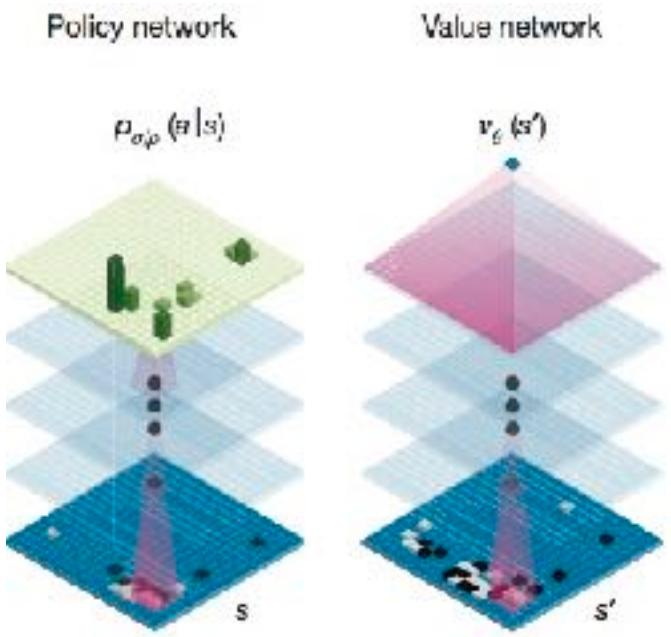
- Learning by trial-and-error
- Reward is often delayed

Computational framework



- At every time step t, the agent perceives the *state* of the environment
- Based on this perception, it chooses an *action*
- The action causes the agent to receive a *numerical reward*
- Find a way of choosing actions, called a *policy* which *maximizes the agent's long-term expected return*

Example: AlphaGo



- Perceptions: state of the board
- Actions: legal moves
- Reward: +1 or -1 at the end of the game
- Trained by playing games against itself
- Invented new ways of playing which seem superior

Basic Principles of Reinforcement Learning

- *All machine learning is driven to minimize prediction errors*
- We use calculus to change parameters for this goal
- In reinforcement learning, the algorithm makes predictions at every time step
- The prediction error is computed between one time step and the next
- We want these predictions to be consistent, i.e. similar to each other
- If *the situation improved since last time step, pick more the last action*

Some definitions

A *policy* $\pi : S \rightarrow A$ is a way of choosing actions

The *value of a state* is the *expected value of a long-term return* (cumulative function of the rewards)

- E.g. average reward per time step over a long horizon
- E.g. Discounted return:

$$V^*(s) = \max_{\pi} \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s]$$

where $\gamma \in [0, 1]$ is a discount factor (probability of the task finishing at each step, or inflation rate) and π dictates the choices of action

General approach: approximate the value of the current policy from data, then use these values to guide policy change

If an action leads to an *improved state of affairs*, the tendency to pick it is strengthened (i.e., the *action is reinforced*)

Temporal-difference learning (Sutton, 1988)

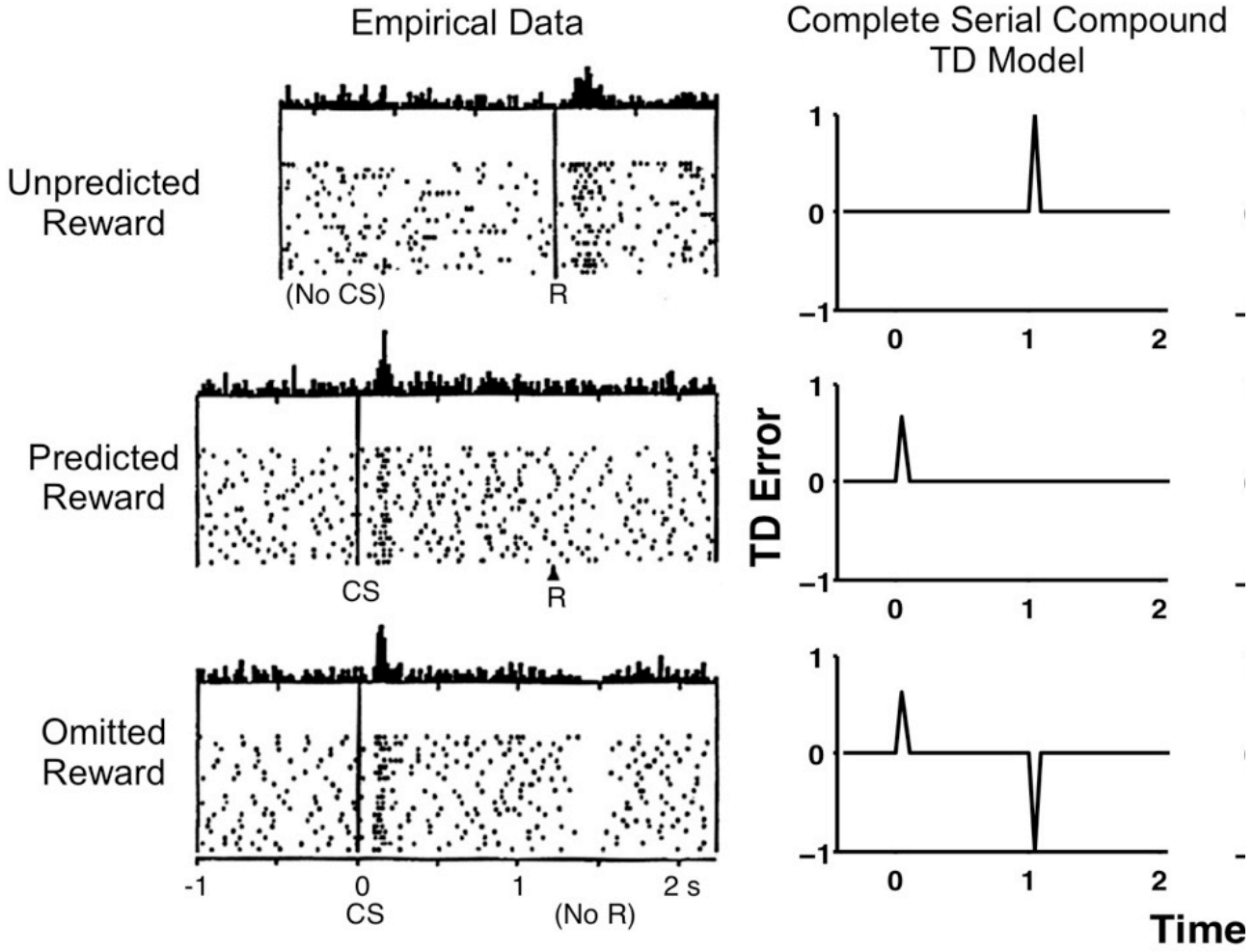
- Estimated value at time t: $V(s_t)$
- Estimated value at time $t+1$: $r_{t+1} + \gamma V(s_{t+1})$
- *Temporal-difference error*:

$$\delta = [r_{t+1} + \gamma V(s_{t+1})] - V(s_t)$$

This is the amount of *surprise* based on the new information at time step $t+1$

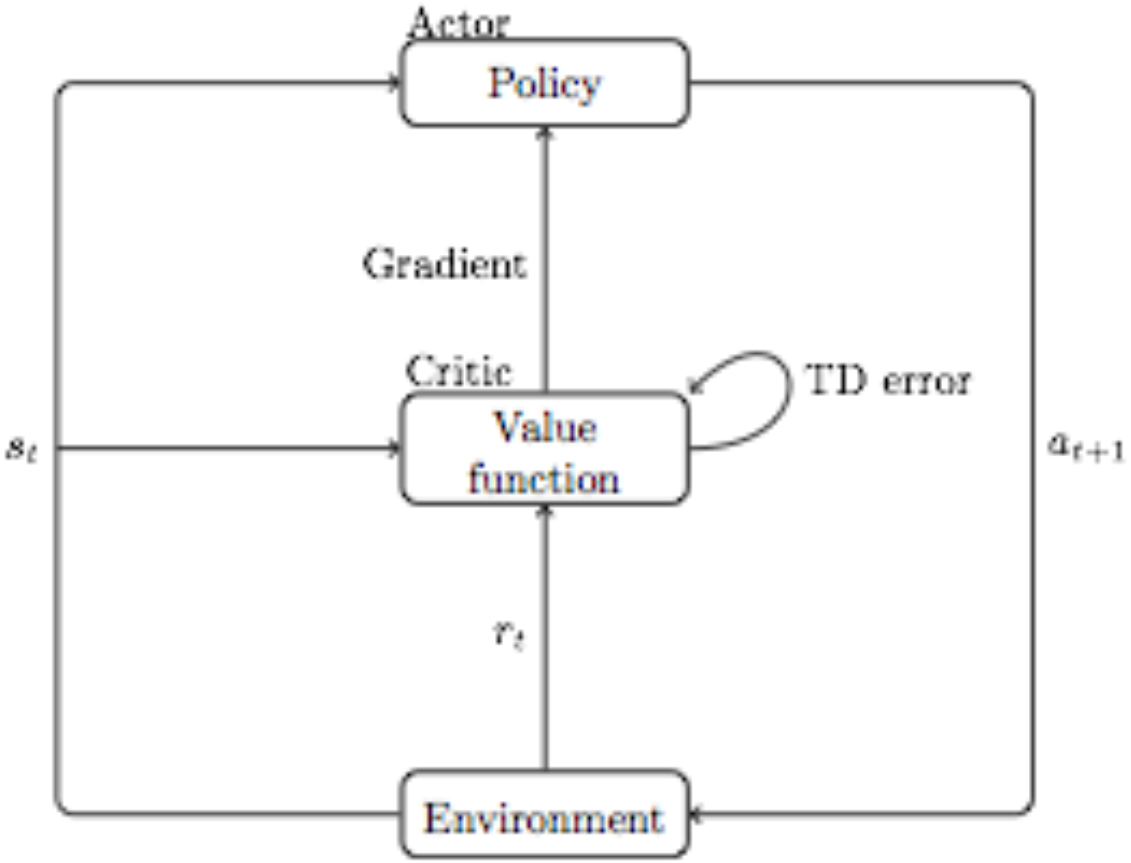
- Main idea: use TD-error to drive the learning of the correct values
- Adapt V such that the expected error becomes 0

Explaining dopaminergic neural activity



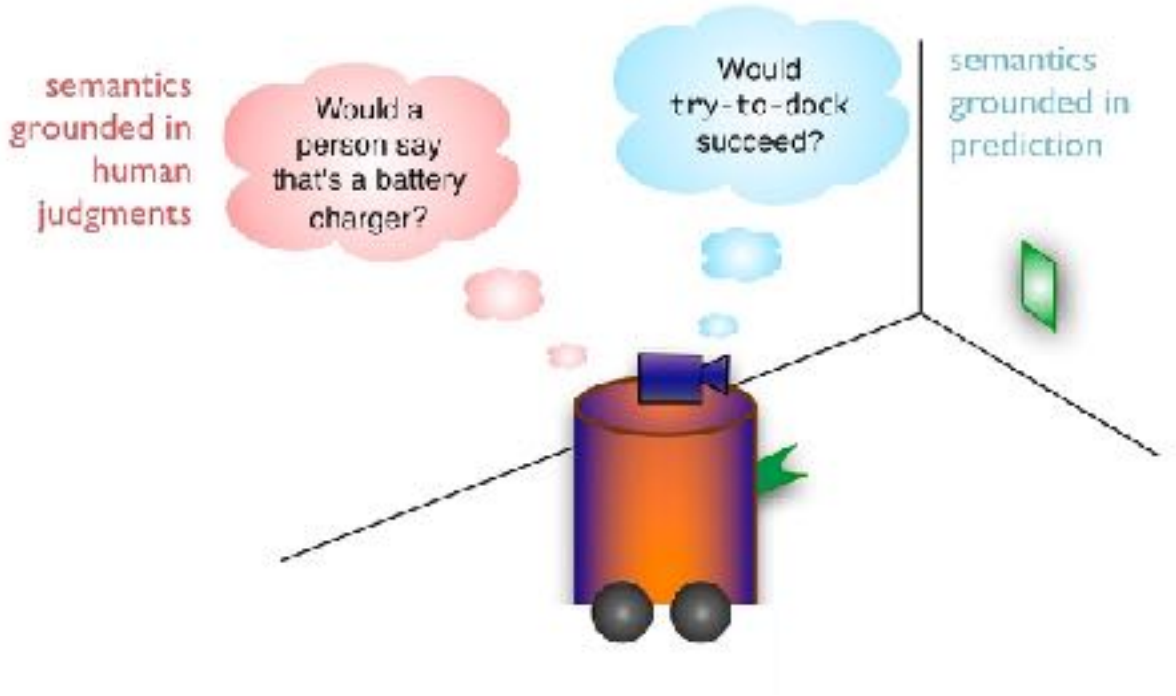
Cf. Shultz et al, 1996 and much current work

Actor-critic architecture



Learning transferrable representations

- How can we use reinforcement learning to build automatically higher-level representations?
- What is the right time scale for actions and computations?



Temporal abstraction

- Consider an activity such as cooking dinner

- High-level steps: choose a recipe, make a grocery list, get groceries, cook,...
- Medium-level steps: get a pot, put ingredients in the pot, stir until smooth, check the recipe ...
- Low-level steps: wrist and arm movement while driving the car, stirring, ...

All have to be seamlessly integrated!

Temporal abstraction in Reinforcement Learning

- Options framework: an option is an extended action that has an initiation condition, an internal way of choosing actions, and a termination condition
- Eg. robot navigation: if there is no obstacle in front (initiation condition) go forward until something is too close (termination condition)
- Learning and planning algorithms work the same way at all levels of abstraction

Options framework

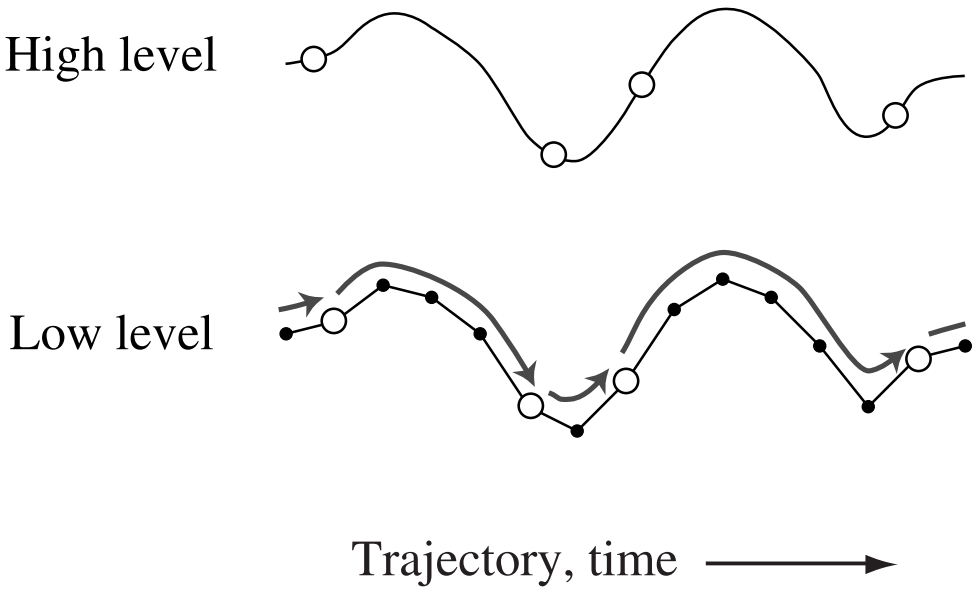
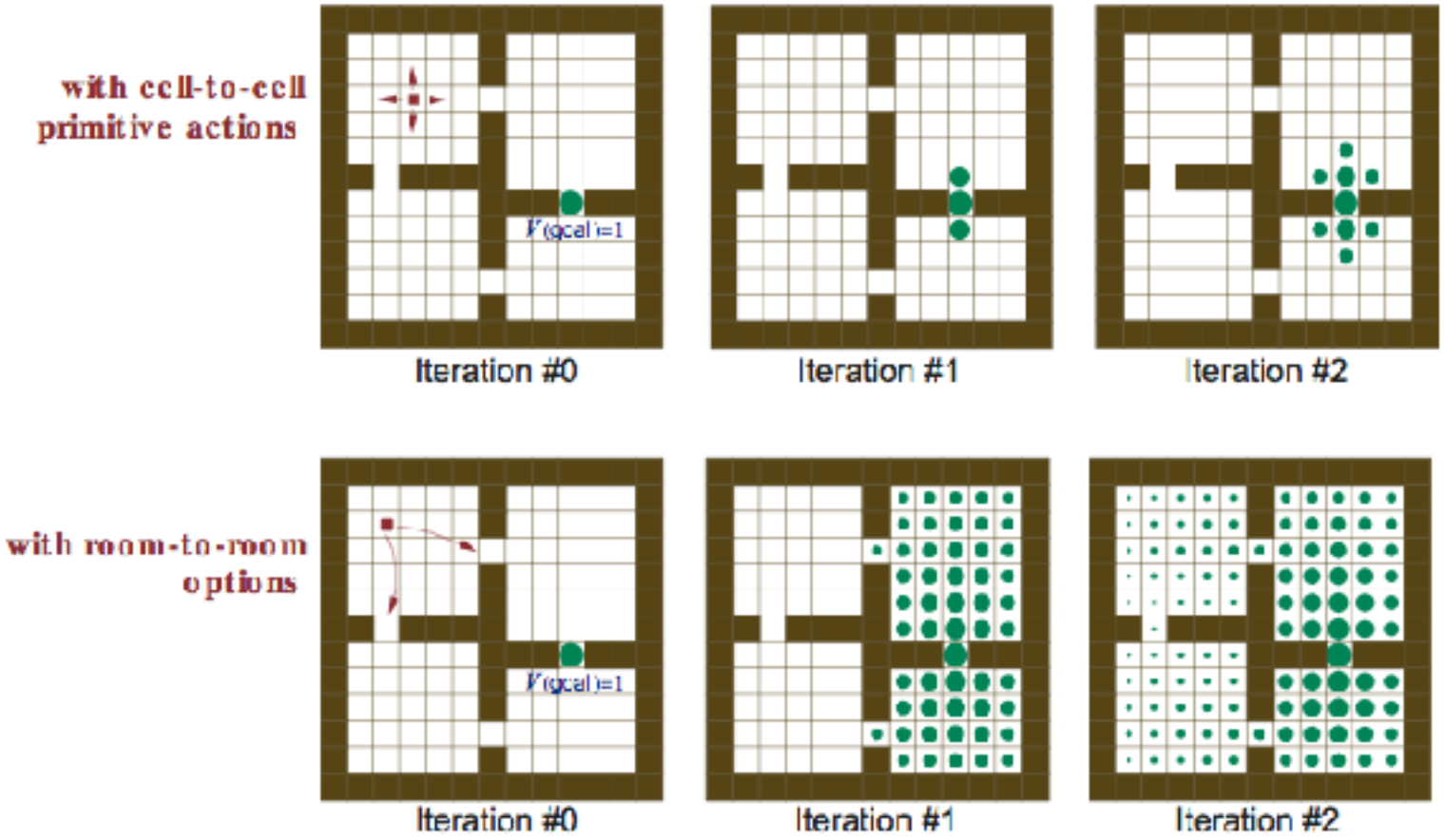


Illustration: Navigation task



Bottleneck states

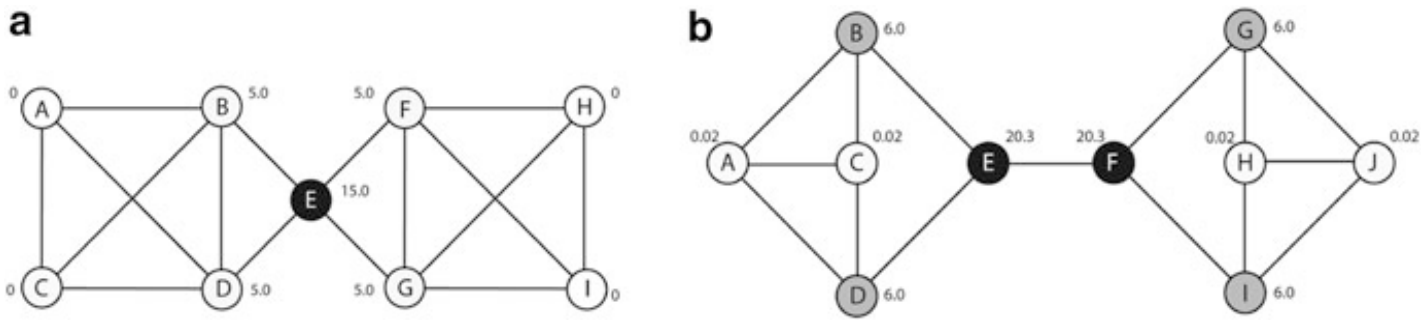
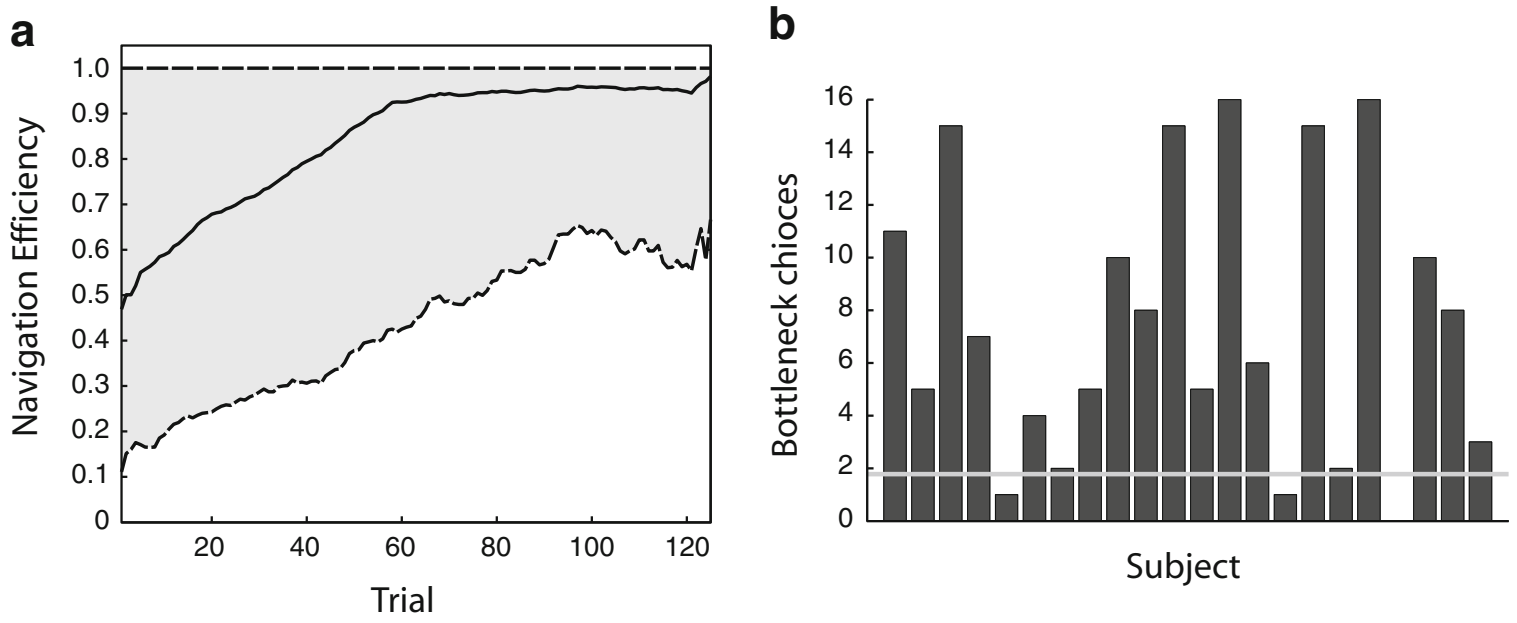


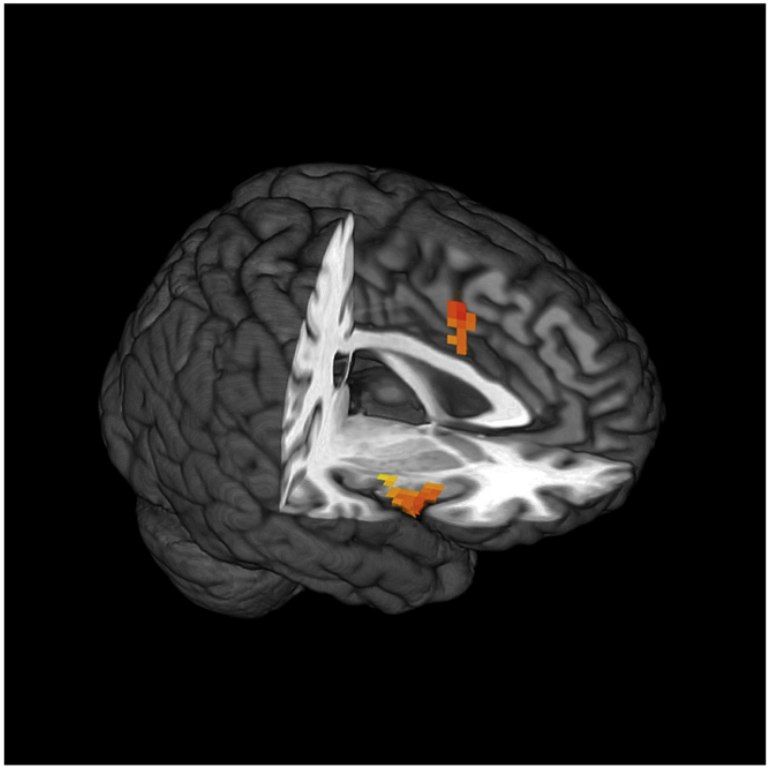
Fig. 5 Graphs underlying the maps of the cities for the first version of the experiment (**a**) and the second one (**b**). Node labels identify the betweenness of each node

Botvinick, Niv et al

People discover bottleneck states

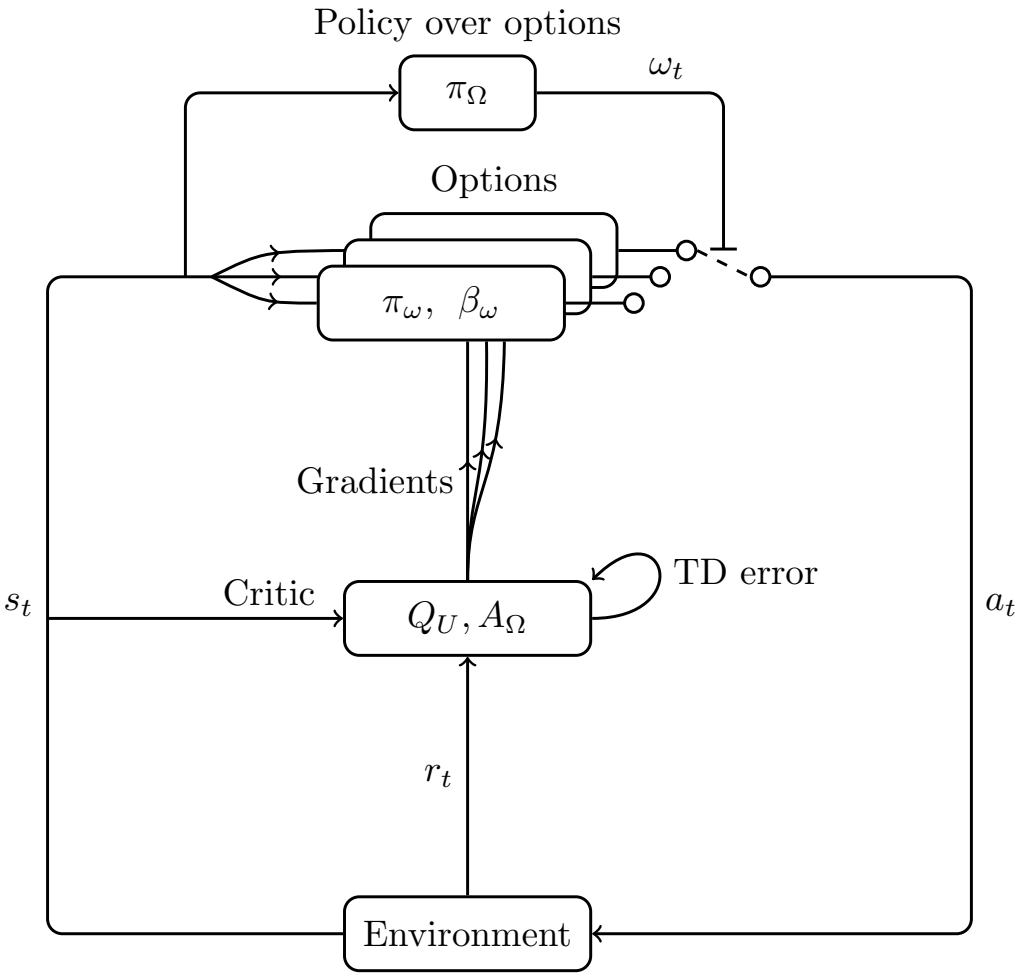


Neural signature for subgoals

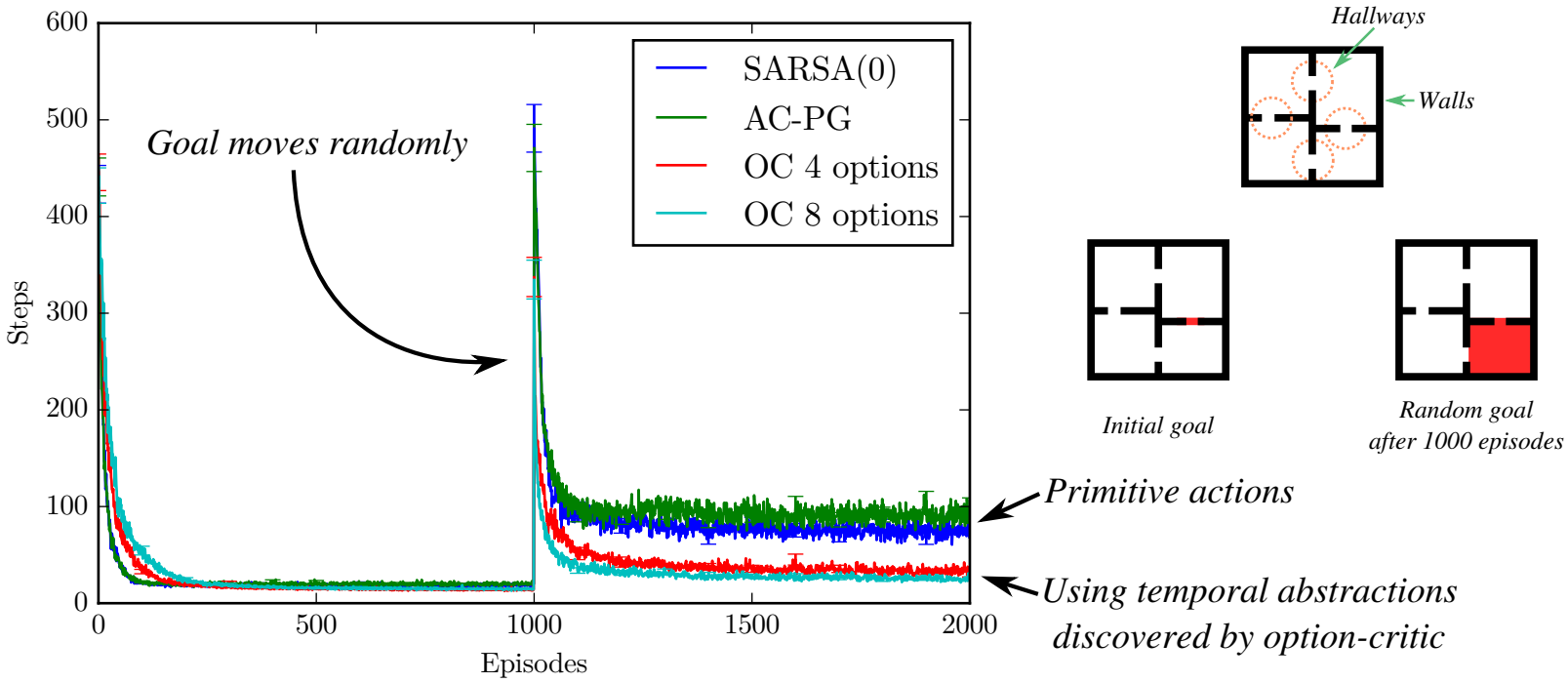


Cf Ribas-Fernandez et al, 2011

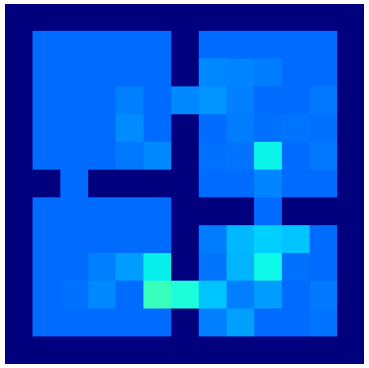
Option-critic architecture



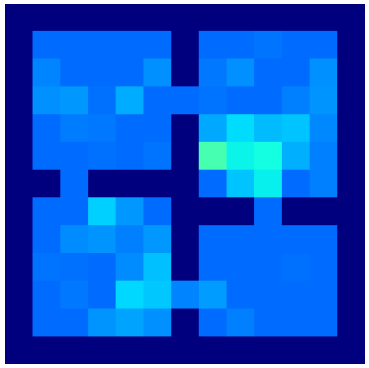
Transfer learning with options



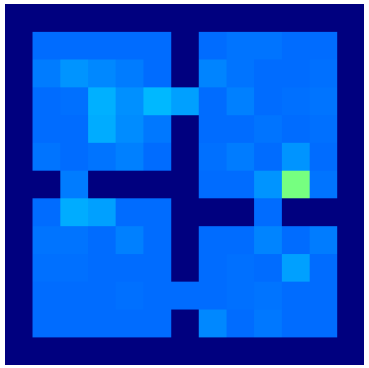
Learned options are intuitive



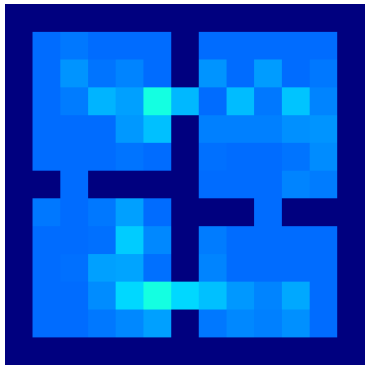
Option 1



Option 2

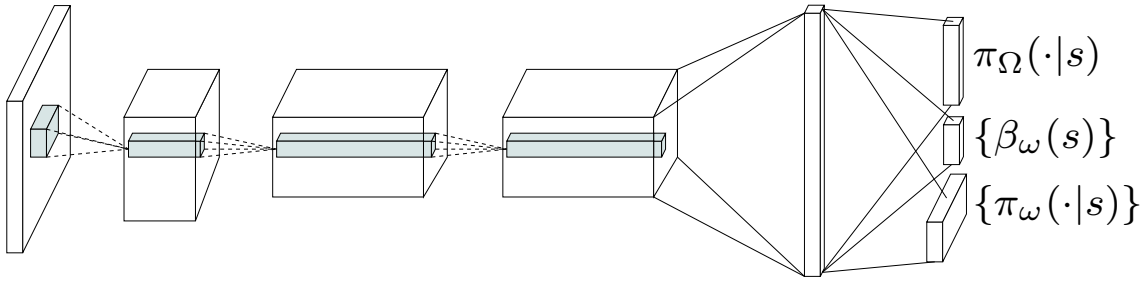


Option 3



Option 4

Deep Learning Option-critic

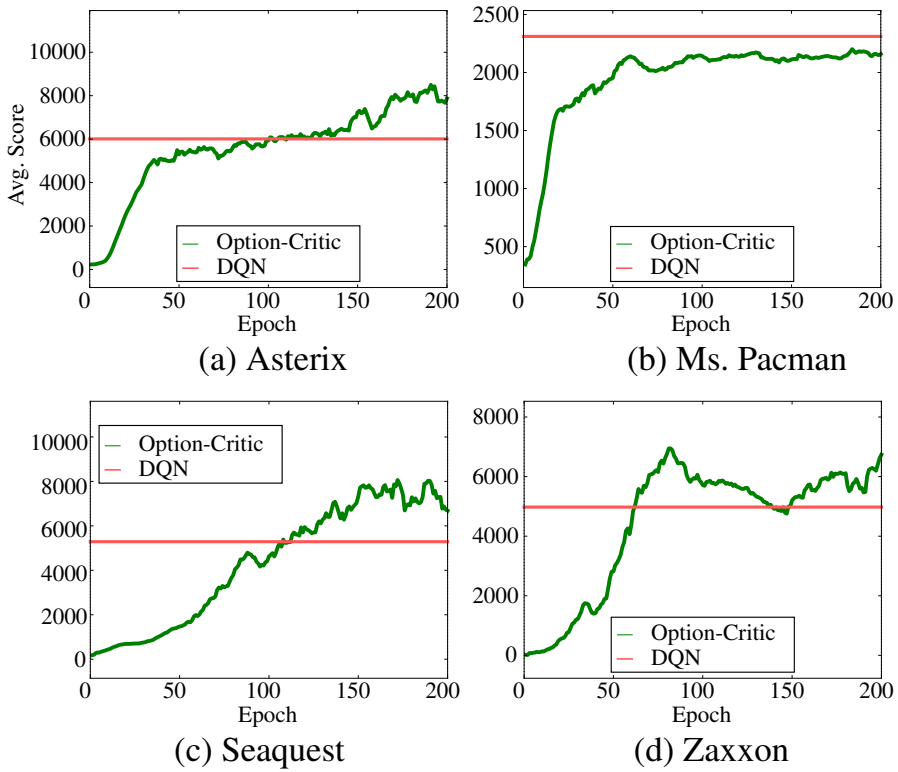


Gradient-based learning:

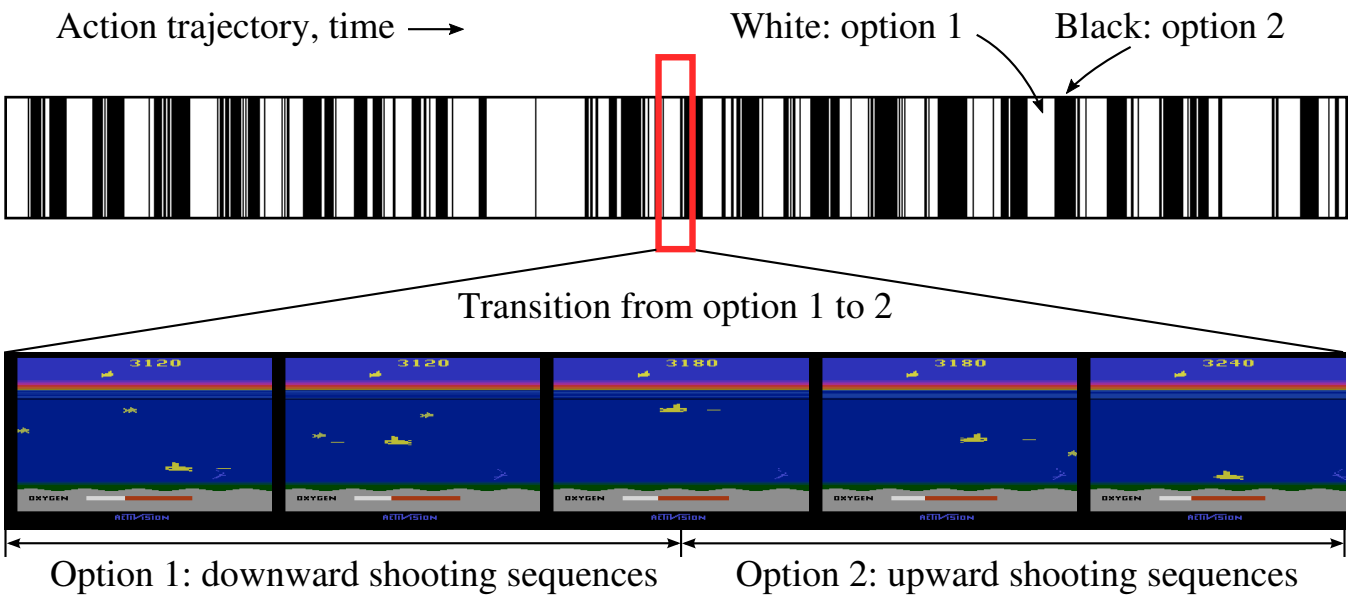
- Optimization criterion: return
- Secondary consideration: avoid switching too often
- Internal policy over options moves to take better actions more often
- Termination conditions shorten if the next option has a large advantage, lengthen otherwise

Results (Atari)

- Results match or exceed those of DQN within a single task

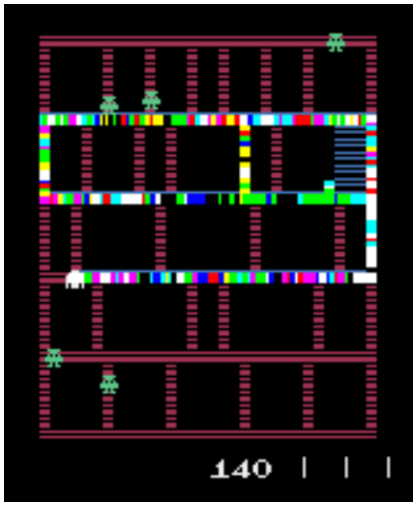


Learned options are intuitive

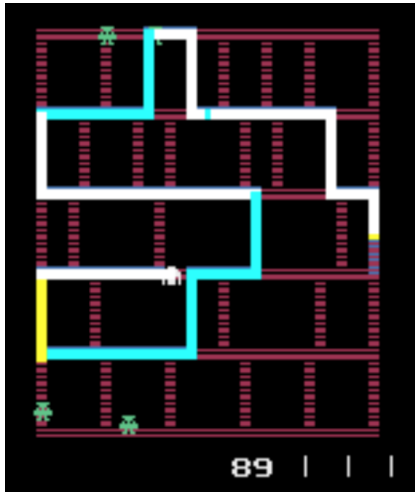


McGill

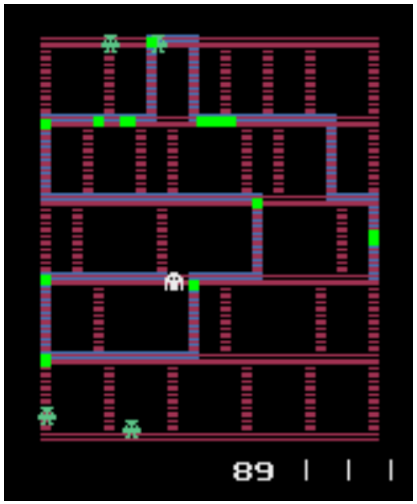
Temporal Abstraction: What is a Good Time Scale for Actions?



(a) Without a deliberation cost, options terminate instantly and are used in any scenario without specialization.



(b) Options are used for extended periods and in specific scenarios through a trajectory, when using a deliberation cost.



(c) Termination is sparse when using the deliberation cost. The agent terminates options at intersections requiring high level decisions.

Summary

- Reinforcement learning is used both to build sophisticated controllers for automated tasks, and to model decision making in the brain (dopamine neutrons)
- To solve large problems, we need temporal abstraction
- Abstractions can be learned automatically from data