Verve: A General Purpose Open Source Reinforcement Learning Toolkit

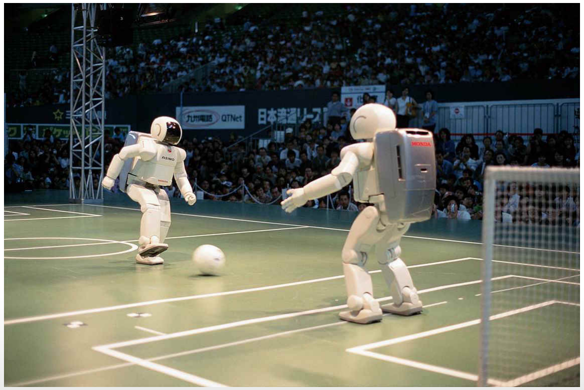
Tyler Streeter, James Oliver, & Adrian Sannier

ASME IDETC & CIE, September 13, 2006

Human Computer Interaction
IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Motivation

Intelligent agents are becoming increasingly important.

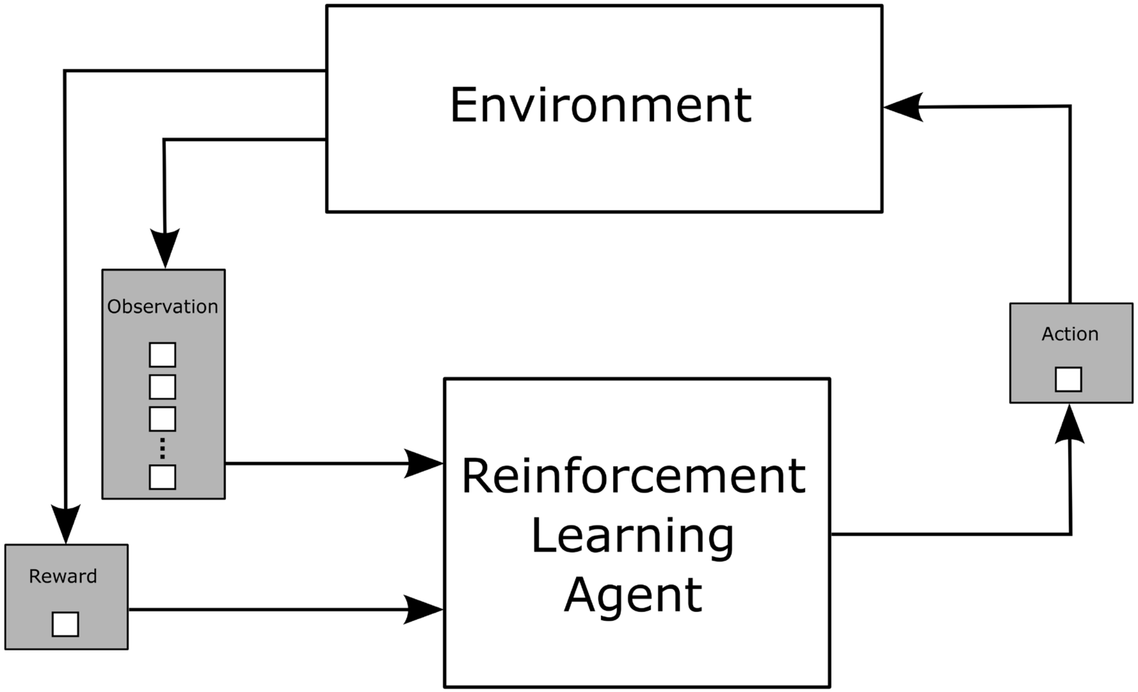


Motivation

- Most intelligent agents today are carefully designed for very specific tasks
- Ideally, we could avoid a lot of work by letting the agents train themselves
- Goal: provide a general purpose agent implementation based on reinforcement learning
- Target audience: Application developers (especially roboticists and game developers)

Reinforcement Learning

- Learning how to behave in order to maximize a numerical reward signal
- Very general: lots of real-world problems can be formulated as reinforcement learning problems

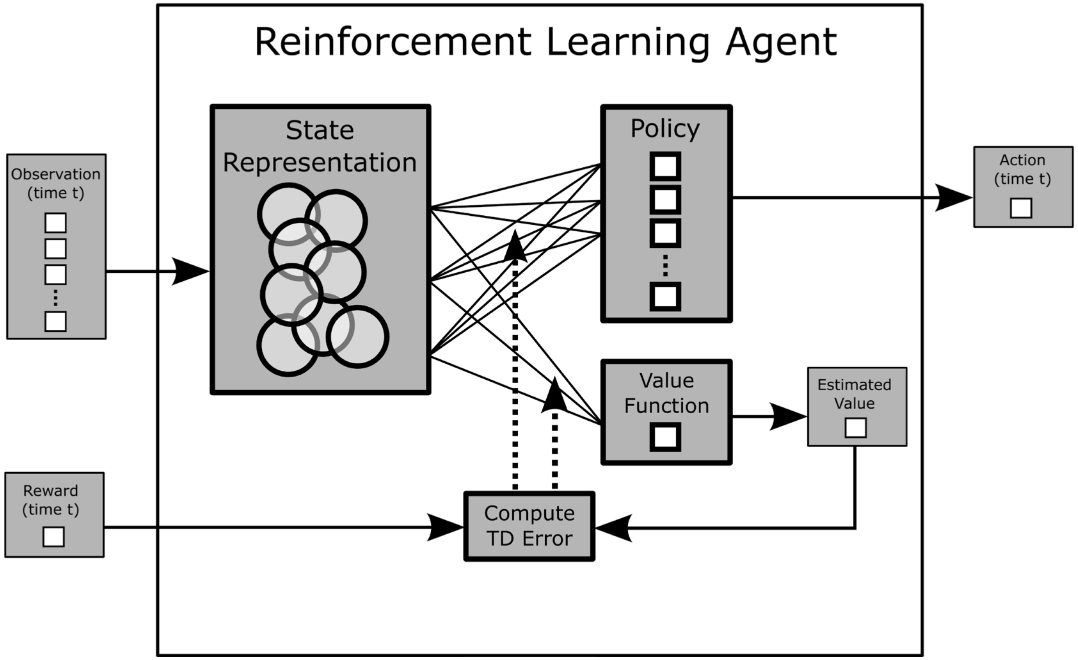


Reinforcement Learning

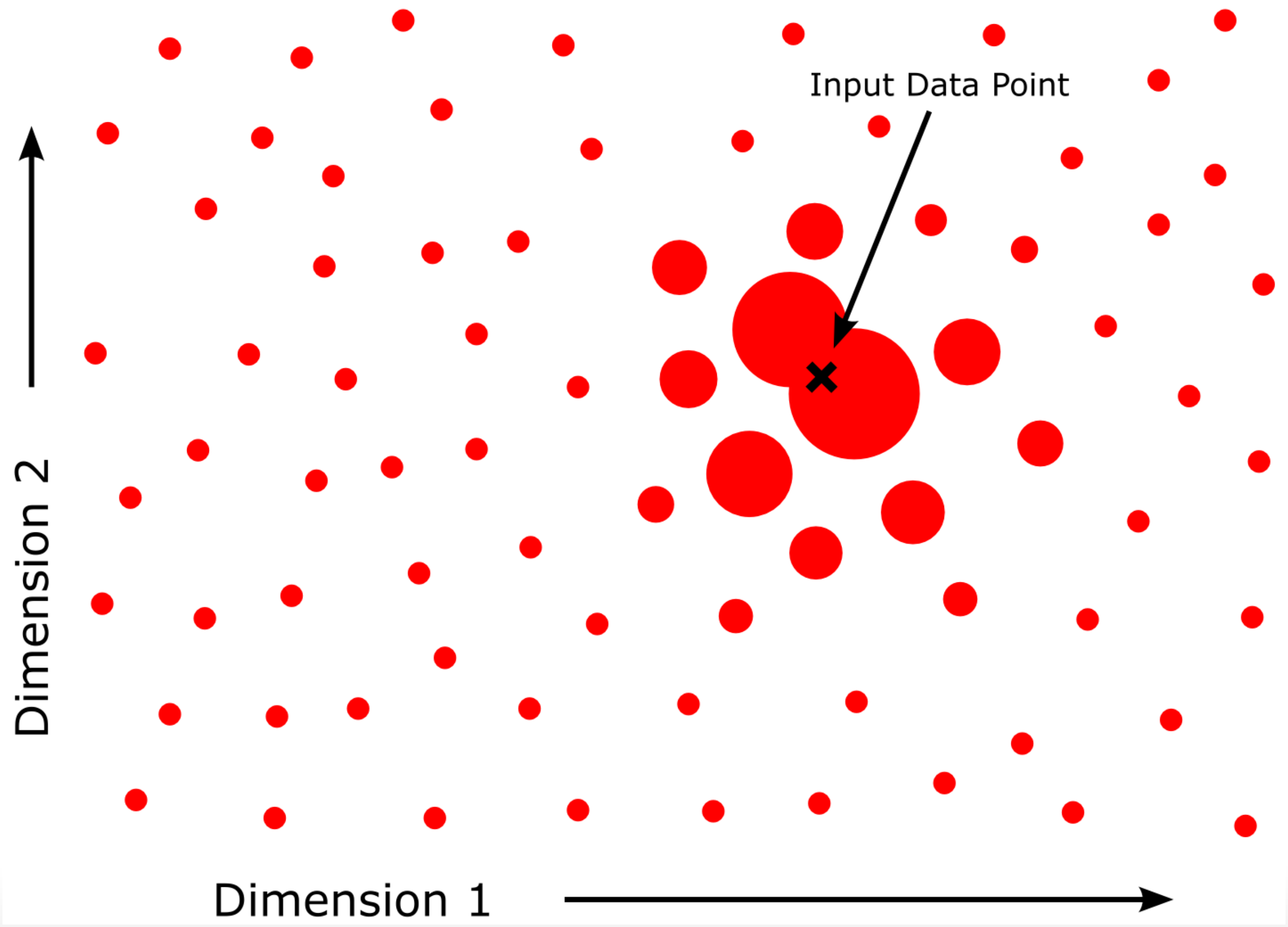
- Typical challenges:
  - Temporal credit assignment
  - Structural credit assignment
  - Exploration vs. exploitation
  - Continuous state spaces
- Solutions:
  - TD learning with value function and policy represented as single-layer neural networks
  - Eligibility traces for connection weights
  - Softmax action selection
  - Function approximation with Gaussian radial basis functions

RL Agent Implementation

- Value function: maps states to “values”
- Policy: maps states to actions
- State representation converts observations to features (allows linear function approximation methods for value function and policy)
- Temporal difference (TD) prediction errors train value function and policy



RBF State Representation



Verve Software Library

- Cross-platform library written in C++ with Python bindings
- License: BSD or LGPL
- Unit tested, heavily-commented source code
- Complete API documentation
- Widely applicable: user-defined sensors, actuators, sensor resolution, and reward function
- Optimized to reduce computational requirements (e.g., dynamically-growing RBF array)

http://verve-agents.sourceforge.net

Free Parameters

- Inputs
  - Number of sensors
  - Choice of discrete or continuous (RBF)
  - Continuous sensor resolution
  - Circular continuous sensors
- Number of outputs
- Reward function
- Agent update rate (step size)
- Learning rates
- Eligibility trace decay time constant
- Reward discounting time constant

C++ Code Sample (1/3)

```
// Define an AgentDescriptor.
verve::AgentDescriptor agentDesc;
agentDesc.addDiscreteSensor(4); // Use 4 possible values.
agentDesc.addContinuousSensor();
agentDesc.addContinuousSensor();
agentDesc.setContinuousSensorResolution(10);
agentDesc.setNumOutputs(3); // Use 3 actions.

// Create the Agent and an Observation initialized to fit this Agent.
verve::Agent agent(agentDesc);
verve::Observation obs;
obs.init(agent);

// Set the initial state of the world.
initEnvironment();
```

C++ Code Sample (2/3)

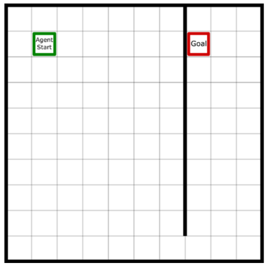
```
// Loop forever (or until some desired learning performance is achieved).
while (1)
{
    // Set the Agent and environment update rate to 10 Hz.
    verve::real dt = 0.1;

    // Update the Observation based on the current state of the world.
    // Each sensor is accessed via an index.
    obs.setDiscreteValue(0, computeDiscreteInput());
    obs.setContinuousValue(0, computeContinuousInput0());
    obs.setContinuousValue(1, computeContinuousInput1());

    // Compute the current reward, which is application-dependent.
    verve::real reward = computeReward();

    // Update the Agent with the Observation and reward.
    unsigned int action = agent.update(reward, obs, dt);
```

C++ Code Sample (3/3)

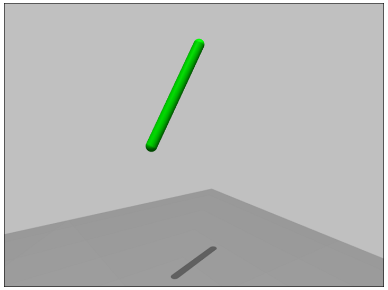
```
    // Apply the chosen action to the environment.
    switch(action)
    {
        case 0:
            performAction0();
            break;
        case 1:
            performAction1();
            break;
        case 2:
            performAction2();
            break;
        default:
            break;
    }

    // Simulate the environment ahead by 'dt' seconds.
    updateEnvironment(dt);
}
```

Examples

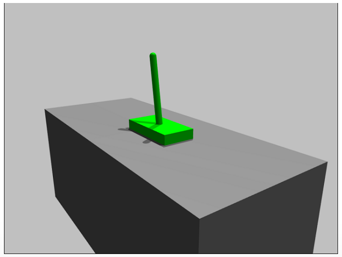
- 2D Maze



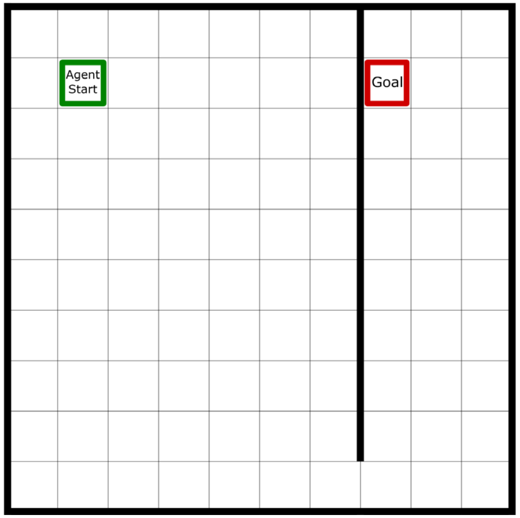
- Pendulum swing-up

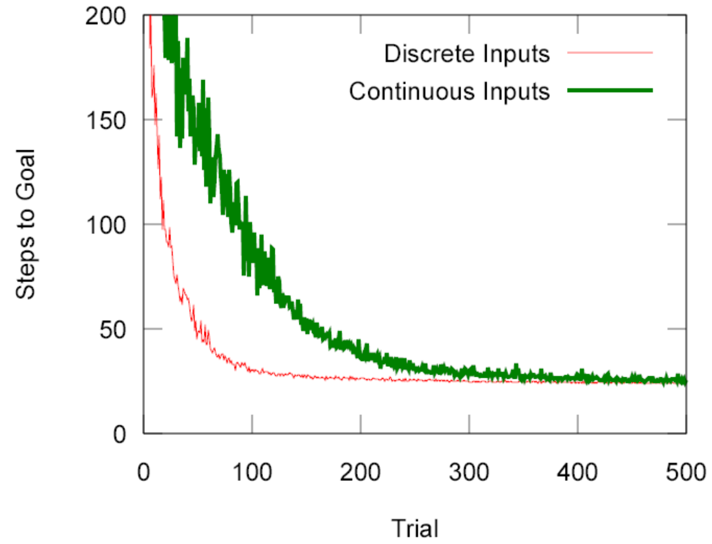


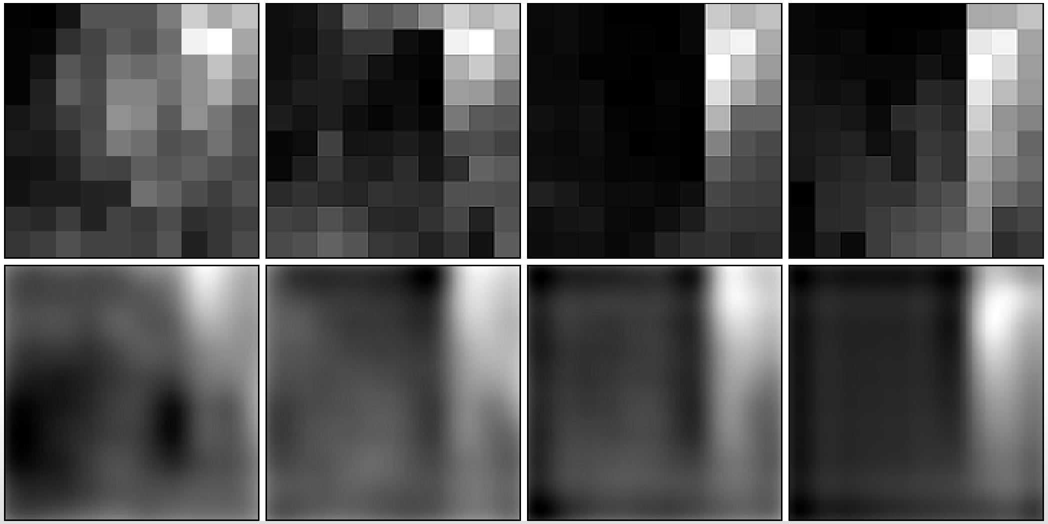
- Cart-pole/inverted pendulum



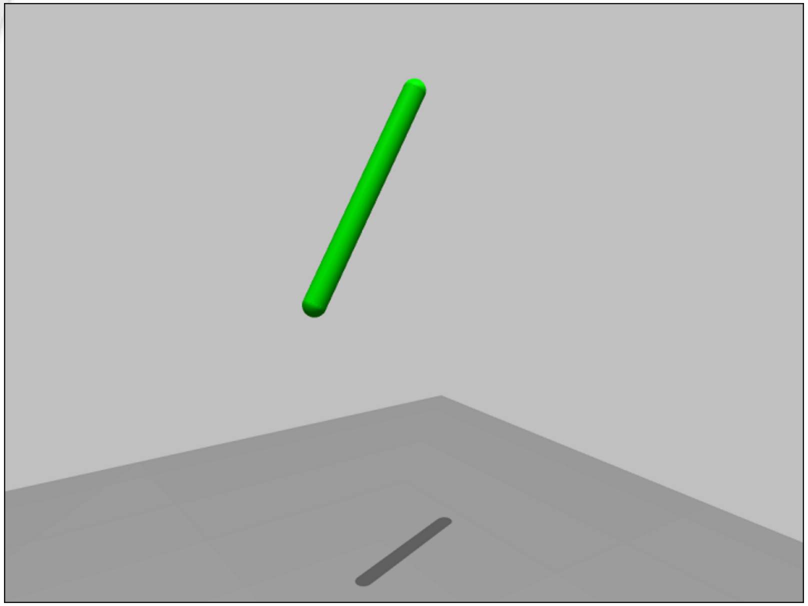
2D Maze Task

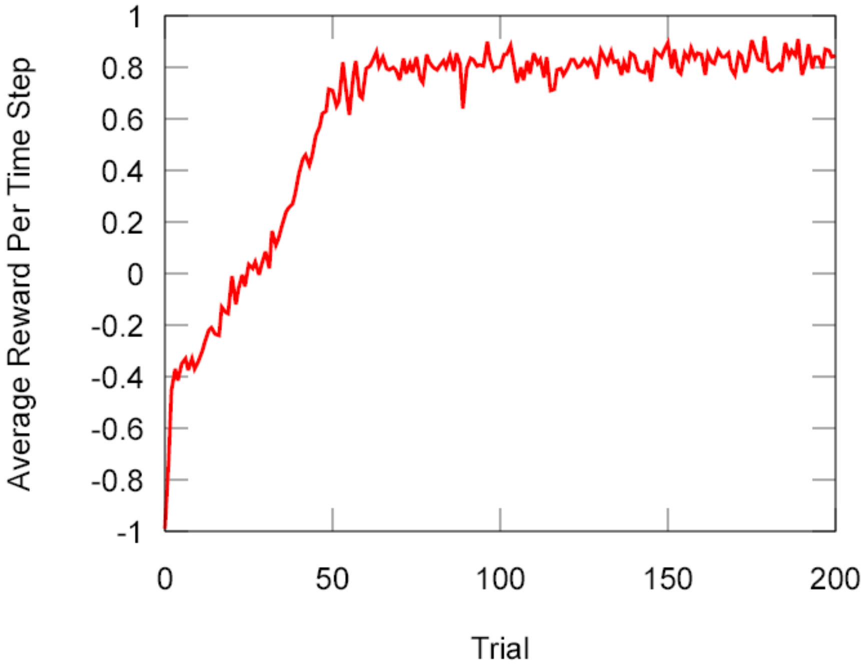

Agent Start
Goal

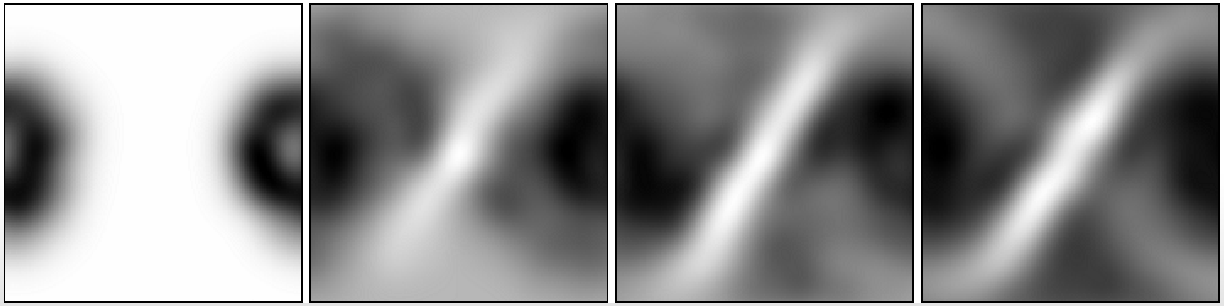

Discrete Inputs
Continuous Inputs
Steps to Goal
Trial
0
50
100
150
200
0
100
200
300
400
500



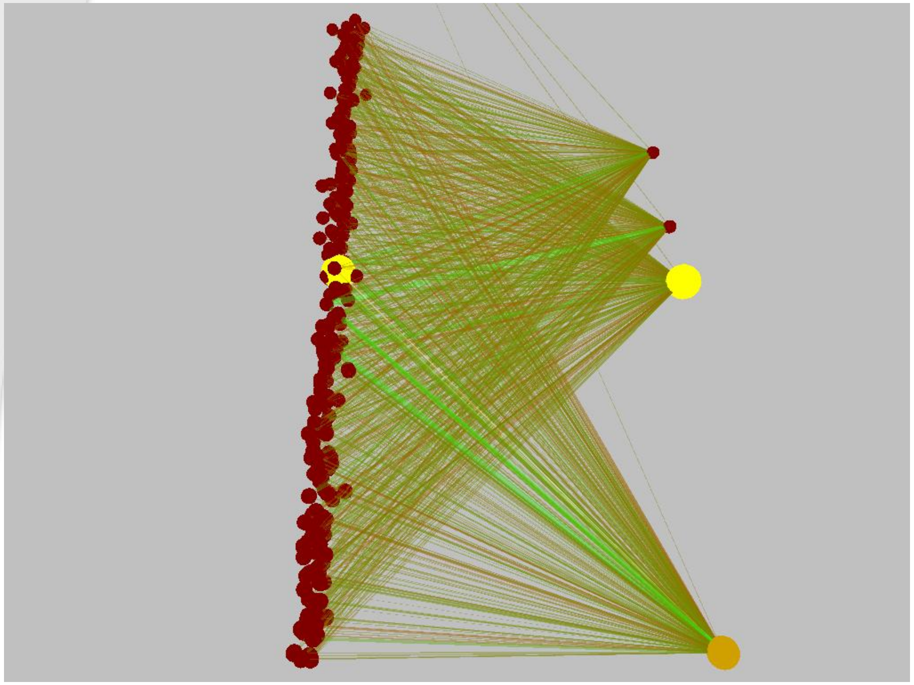
Pendulum Swing-Up Task

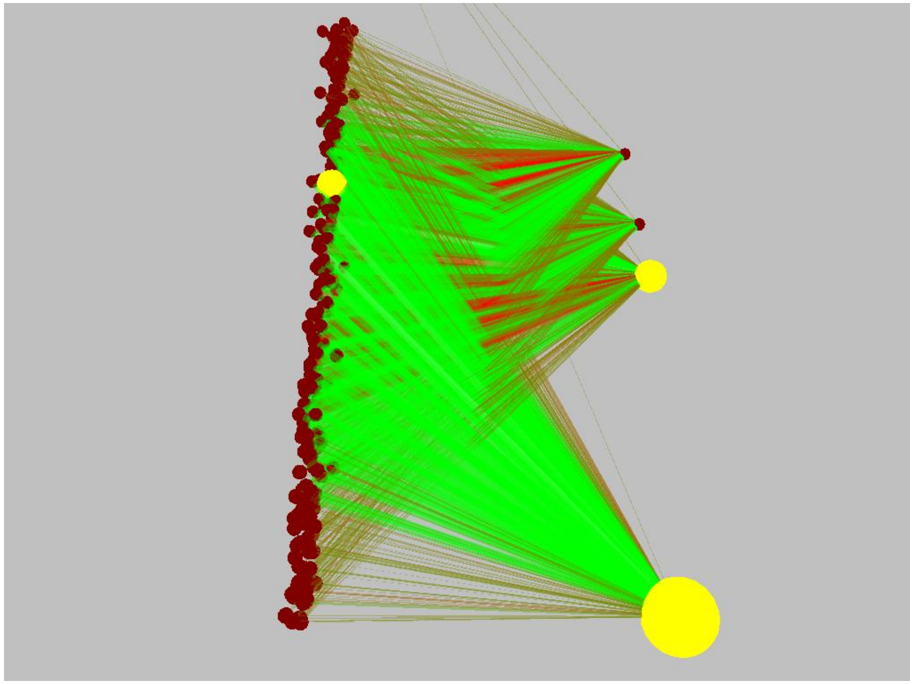




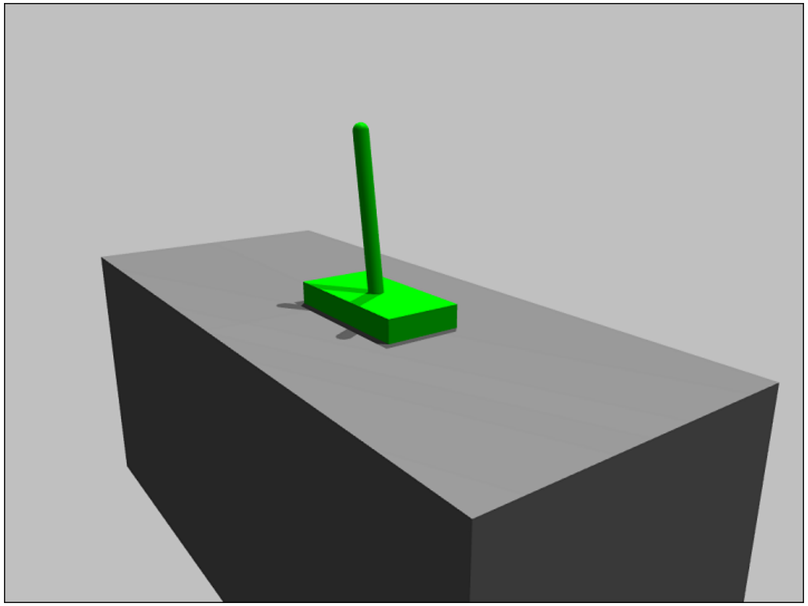


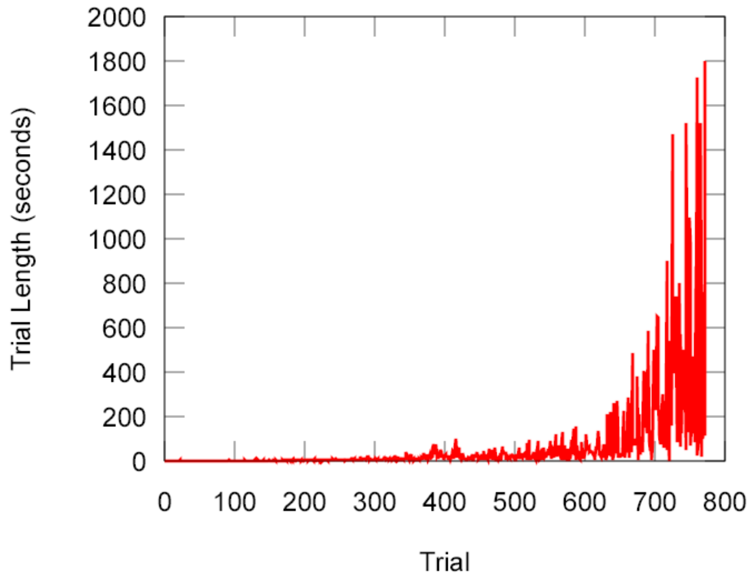
Pendulum Neural Networks





Cart-Pole/Inverted Pendulum Task

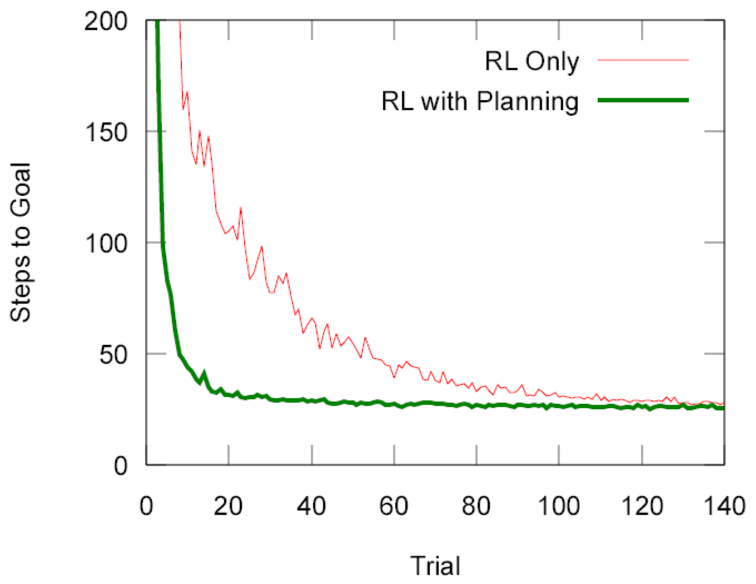




Experimental Feature - Planning

- Planning: training the value function and policy from a learned model of the environment (i.e. reinforcement learning from simulated experiences)
- Reduces training time significantly

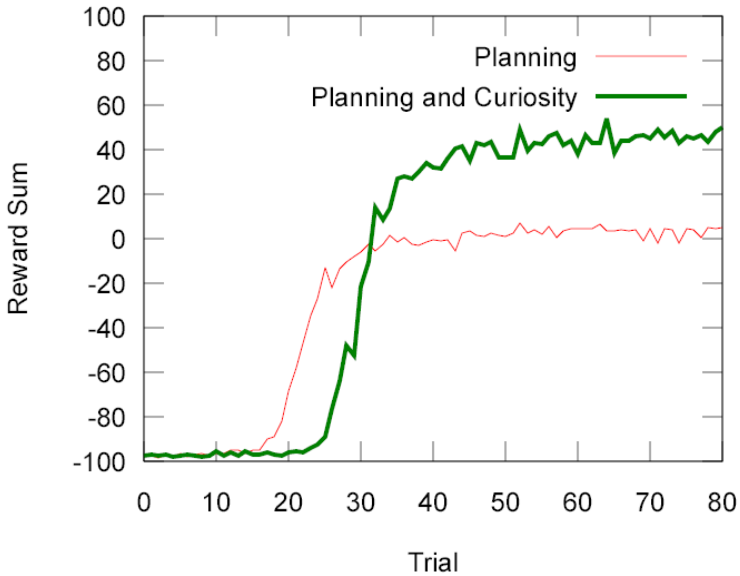
2D Maze Task with Planning



Experimental Feature - Curiosity

- Curiosity: an intrinsic drive to explore unfamiliar states
- Provide extra rewards proportional to uncertainty or "learning progress"
- Drives agents to improve mental models of the environment (used for planning)

Multiple Rewards Task with Curiosity



Future Work

- The exhaustive RBF state representation is too slow for high-dimensional state spaces. Possible solutions: dimensionality reduction (e.g., using PCA or ICA), hierarchical state and action representations, and focused attention
- Temporal state representation (e.g., tapped delay lines)