Introduction to Reinforcement Learning

Lecture 1

Subir Varma

Class Information

- Class Time:
  Sat, Feb 10: 10AM – 12 Noon, 2PM – 4PM
  Sun, Feb 11: 10AM – 12 Noon
  Mon, Feb 12 – Fri, Feb 16: 6:30PM – 8:30PM
  Sat, Feb 17: 10AM – 12 Noon, 2PM – 4PM
- Classroom: Academic Block 3, FF216
- Lectures available at Website:
  https://subirvarma.github.io/GeneralCognitics/Courses.html
- Contact Information: subir.varma@iitgn.ac.in

Book

"Reinforcement Learning: An Introduction" by Richard Sutton and Andrew Bartow.

2nd Edition: Available online at:
http://incompleteideas.net

Pre-Requisites

Knowledge of

- Introductory Machine Learning
- Basic Probability Theory, Markov Chains
- High school level Calculus (Partial Differentiation)

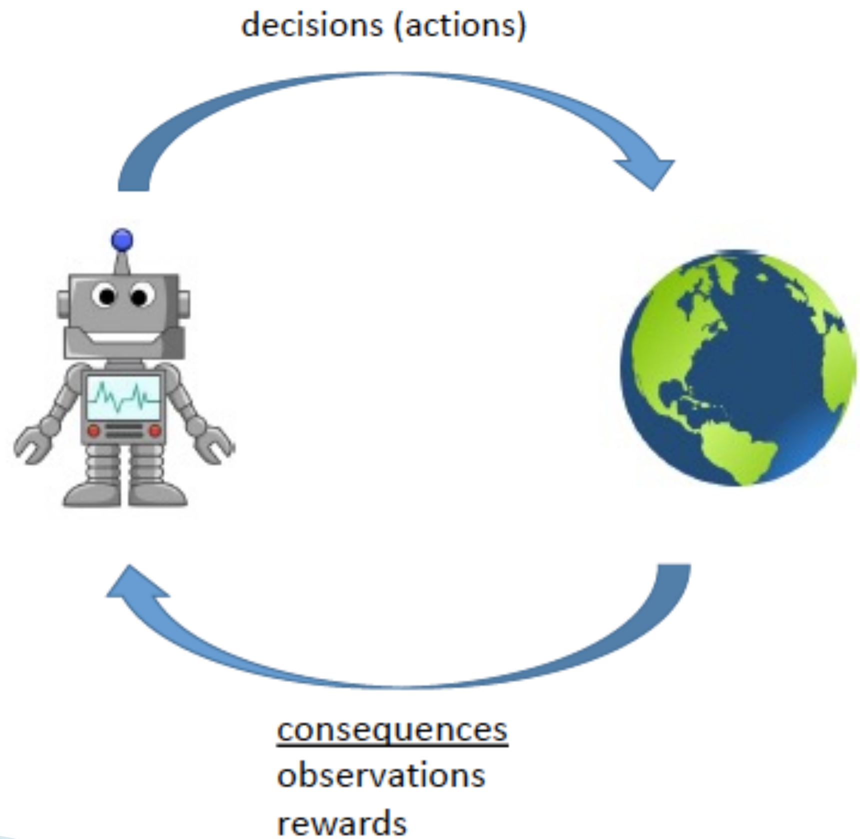
Software Knowledge:

- Python Programming
- Keras, PyTorch

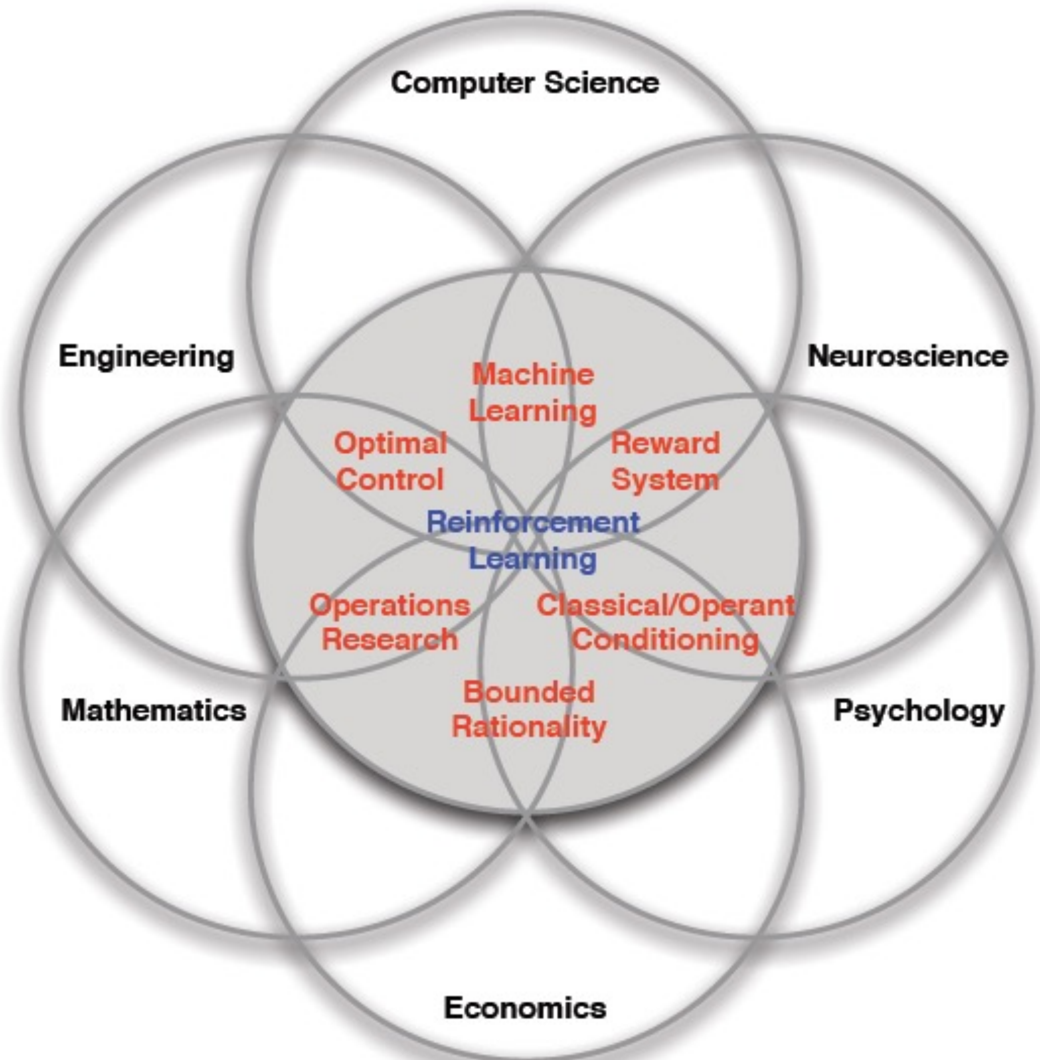
What is Reinforcement Learning?

Science of Making Decisions

By Interacting with the Environment



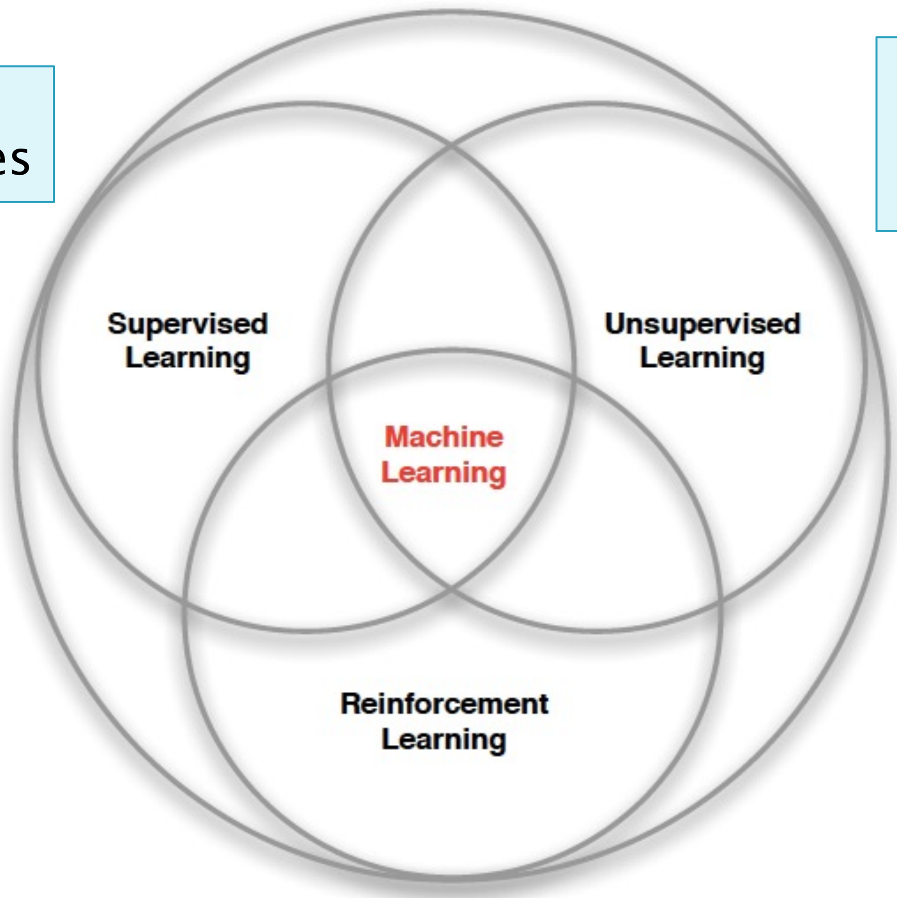
Many Faces of Reinforcement Learning



Branches of Machine Learning

Training Using Labeled Examples

Detection of Patterns in Unstructured Data



Training Using Rewards

Characteristics of RL

What makes reinforcement learning different from other machine learning paradigms?

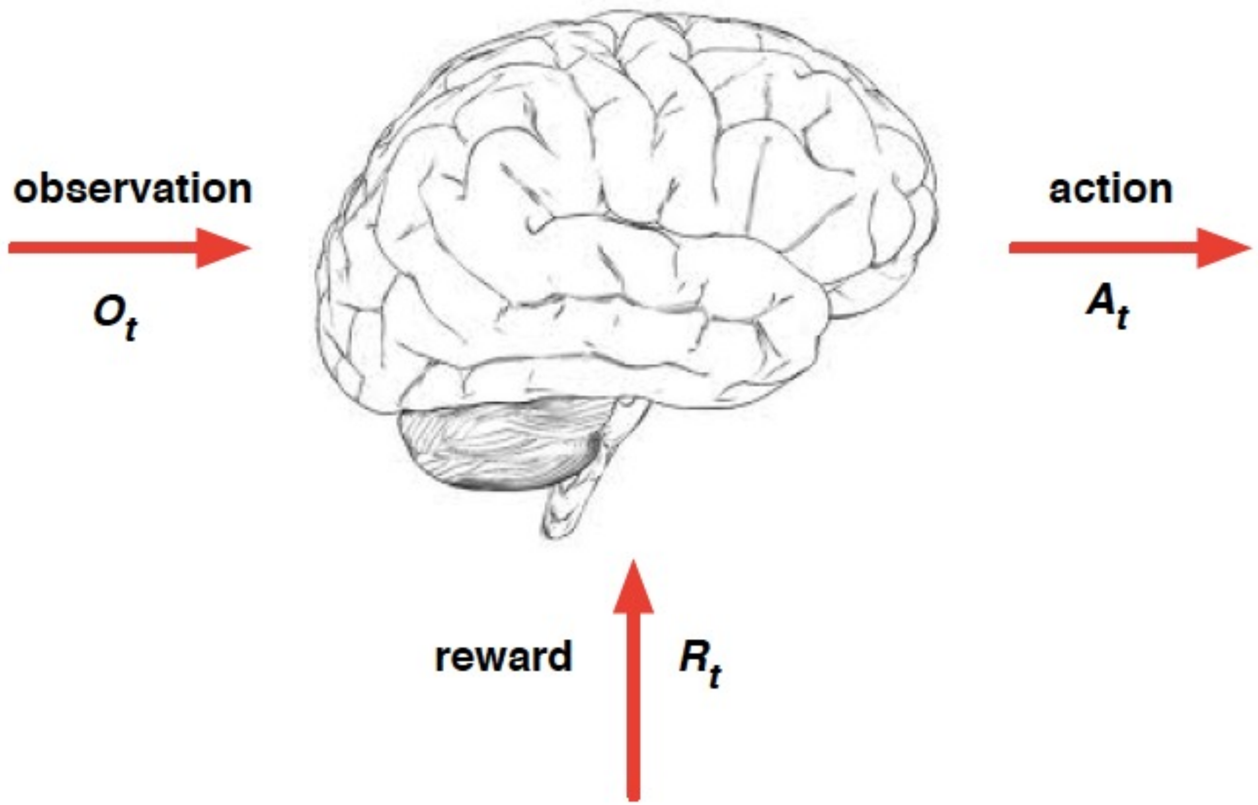
- There is no supervisor, only a *reward* signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

Examples of Reinforcement Learning

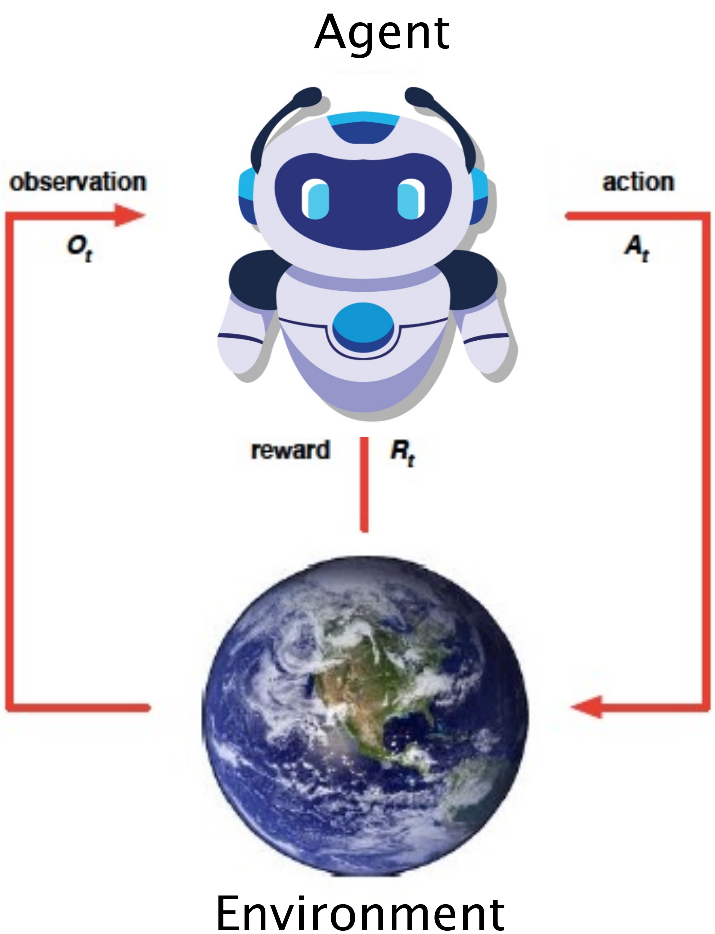
- Playing Video Games such as Atari, Go or Chess
- Training an LLM: Reinforcement Learning based on Human Feedback (RLHF)
- Optimizing Online Ads
- Making a Robot walk
- Managing an Investment Portfolio

The RL Problem: Agent and Environment

Agent



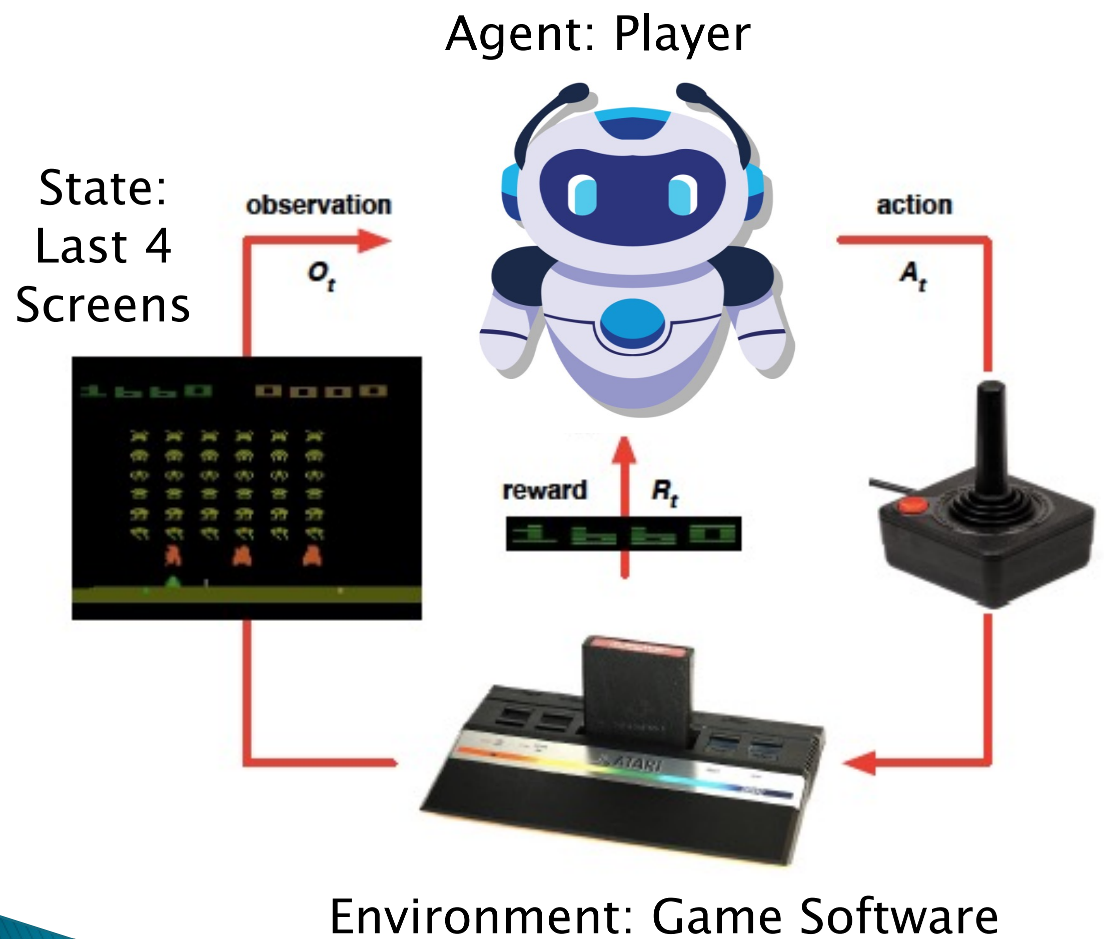
Agent and Environment



- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at env. step

Agent has no control over the Environment's response

Atari Example



- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores

Examples

Actions: muscle contractions
Observations: sight, smell
Rewards: food

Actions: motor current or torque
Observations: camera images
Rewards: task success measure (e.g., running speed)

Actions: what to purchase
Observations: inventory levels
Rewards: profit

Playing Atari with RL

[Playing Atari Breakout](https://www.youtube.com/watch?v=V1eYniJ0Rnk&vl=en)

https://www.youtube.com/watch?v=V1eYniJ0Rnk&vl=en

The RL Problem: Rewards

Rewards

- A reward R_t is a scalar feedback signal
- Indicates how well agent is doing at step t
- The agent's job is to maximise cumulative reward

Reinforcement learning is based on the reward hypothesis

Definition (Reward Hypothesis)

All goals can be described by the maximisation of expected cumulative reward

Examples of Rewards

- Defeat the world champion at Backgammon
  - +/−ve reward for winning/losing a game
- Play many different Atari games better than humans
  - +/−ve reward for increasing/decreasing score
- Manage an investment portfolio
  - +ve reward for each $ in bank
- Control a power station
  - +ve reward for producing power
  - −ve reward for exceeding safety thresholds
- Make a humanoid robot walk
  - +ve reward for forward motion
  - −ve reward for falling over

Sequential Decision Making

- Goal: *select actions to maximise total future reward*
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward

- Examples:
  - A financial investment (may take months to mature)
  - Refuelling a helicopter (might prevent a crash in several hours)
  - Blocking opponent moves (might help winning chances many moves from now)

The RL Problem: State

History and State

- The history is the sequence of observations, actions, rewards

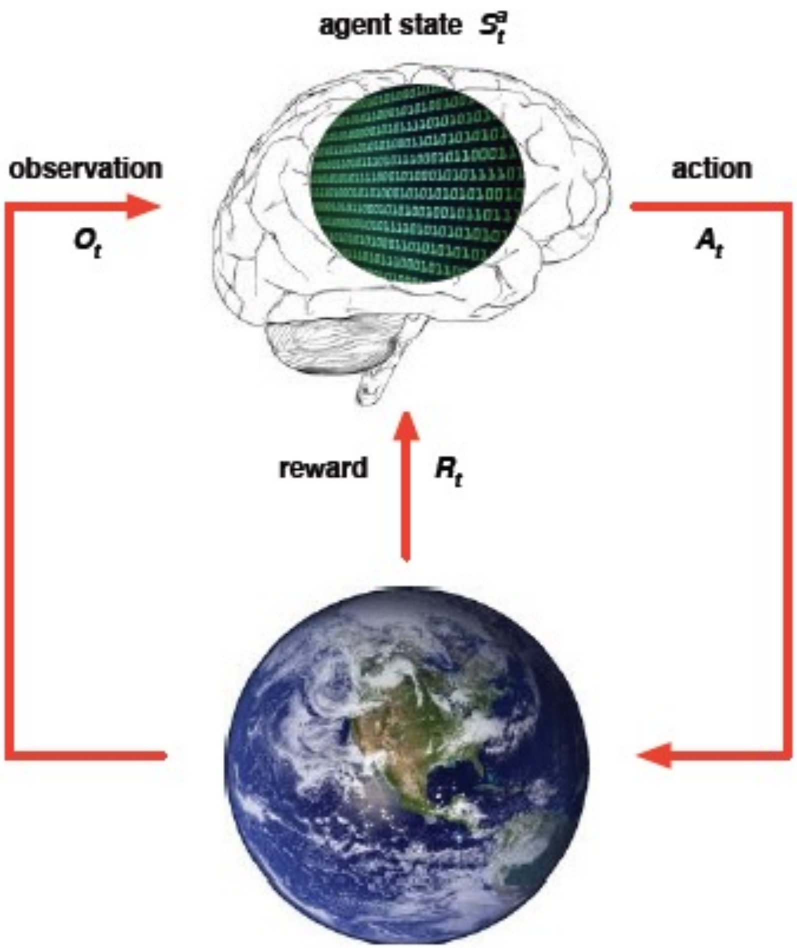
$$H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t$$

- i.e. all observable variables up to time t
- i.e. the sensorimotor stream of a robot or embodied agent
- What happens next depends on the history:
  - The agent selects actions
  - The environment selects observations/rewards

- State is the information used to determine what happens next
- Formally, state is a function of the history:

$$S_t = f(H_t)$$

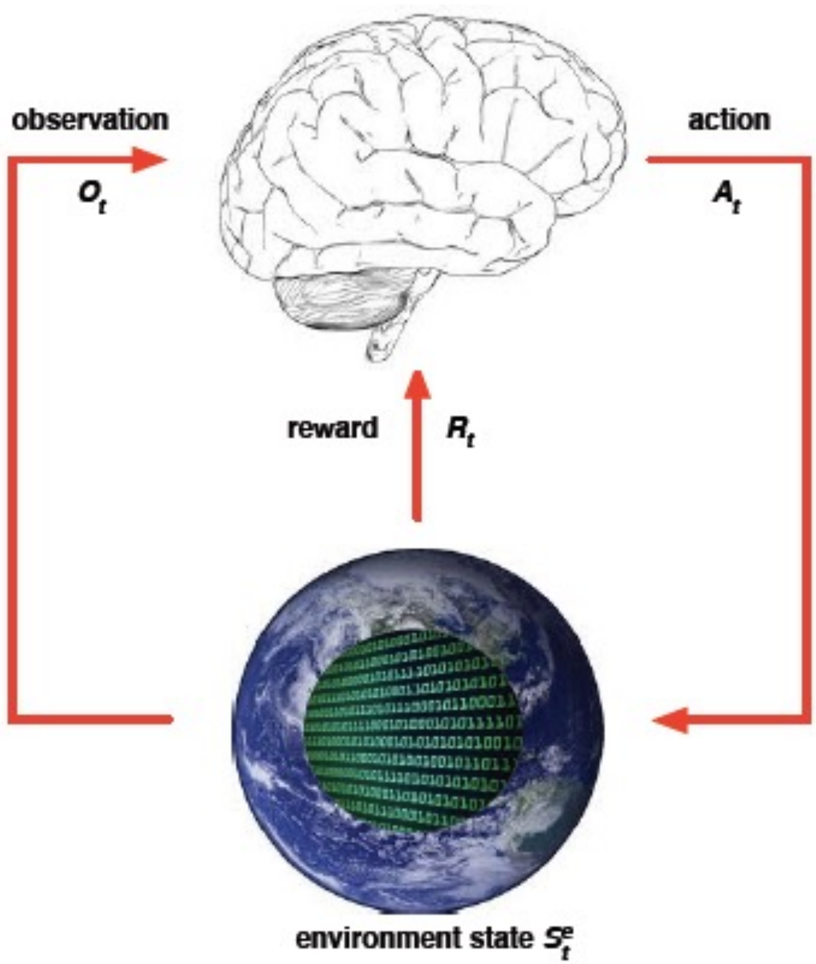
Agent State



- The agent state S_t^a is the agent's internal representation
- i.e. whatever information the agent uses to pick the next action
  - i.e. it is the information used by reinforcement learning algorithms
  - It can be any function of history:

$$S_t^a = f(H_t)$$

Environment State



- The environment state S_t^e is the environment's private representation
- i.e. whatever data the environment uses to pick the next observation/reward

- The environment state is not usually visible to the agent
- Even if S_t^e is visible, it may contain irrelevant information

An Useful Property: Markov State

An information state (a.k.a. Markov state) contains all useful information from the history.

Definition

A state S_t is Markov if and only if

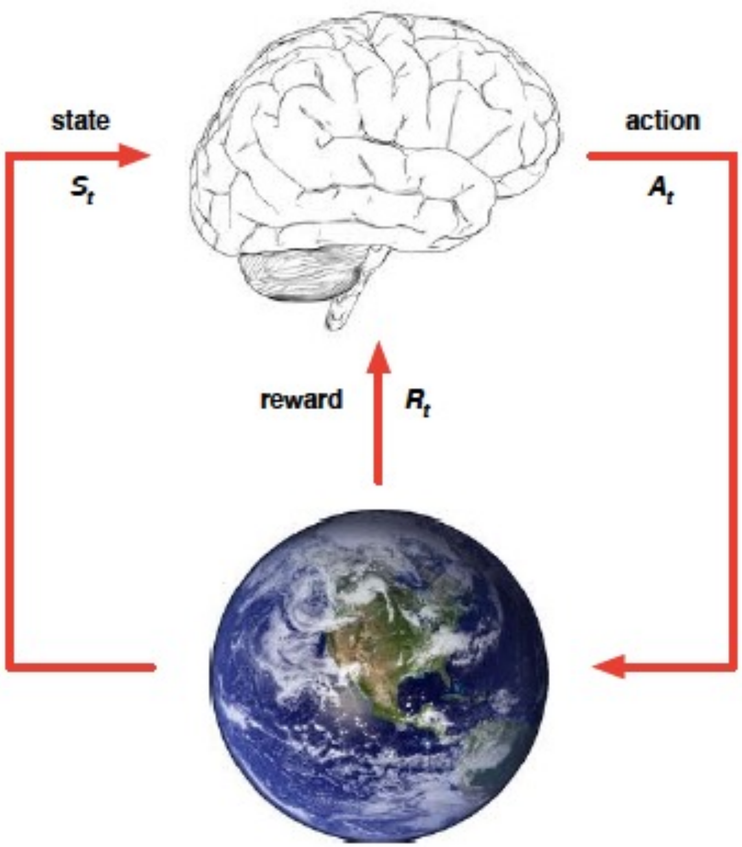
$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$$

- "The future is independent of the past given the present"

$$H_{1:t} \rightarrow S_t \rightarrow H_{t+1:\infty}$$

- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future
- The environment state S_t^e is Markov
- The history H_t is Markov

Fully Observable Environments



Full observability: agent directly observes environment state

$$O_t = S_t^a = S_t^e$$

- Agent state = environment state = information state
- Formally, this is a Markov decision process (MDP)
- (Next lecture and the majority of this course)

Partially Observable Environments

- Partial observability: agent indirectly observes environment:
  - A robot with camera vision isn't told its absolute location
  - A trading agent only observes current prices
  - A poker playing agent only observes public cards
- Now agent state $\neq$ environment state
- Formally this is a partially observable Markov decision process (POMDP)

- Agent must construct its own state representation S_t^a, e.g.
  - Complete history: $S_t^a = H_t$
  - Beliefs of environment state: $S_t^a = (\mathbb{P}[S_t^e = s^1], ..., \mathbb{P}[S_t^e = s^n])$
  - Recurrent neural network: $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$

Components of an RL Agent

RL Agent Components

- An RL agent may include one or more of these components:
  - Policy: agent's behaviour function
  - Value function: how good is each state and/or action
  - Model: agent's representation of the environment

Policy

- A policy is the agent's behaviour
- It is a map from state to action,

- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

Value Function

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \right]$$

A Value Function specifies what is good in the long run

It is better to make decisions on the basis of Value Functions rather than Immediate Rewards

Model

- A model predicts what the environment will do next
- $\mathcal{P}$ predicts the next state
- $\mathcal{R}$ predicts the next (immediate) reward, e.g.

$$\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$
$$\mathcal{R}_{s}^{a} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$

The Agent's Representation of the Environment

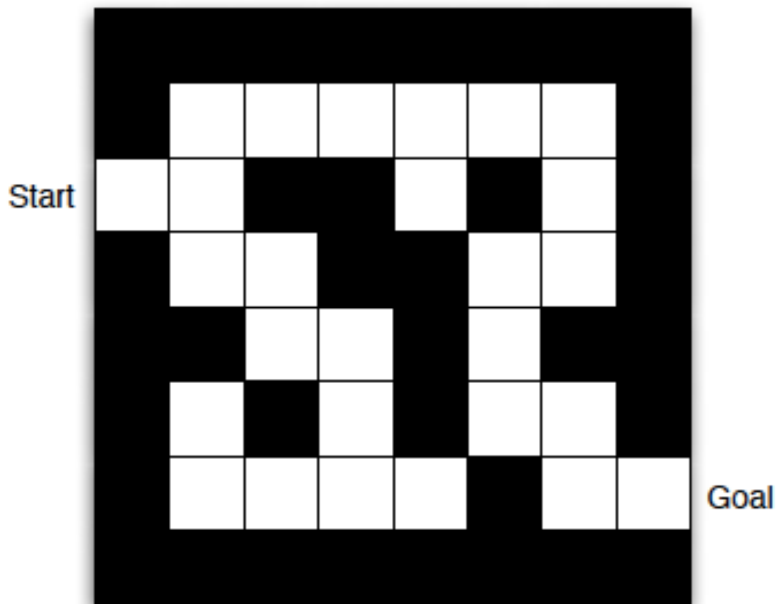
Central Problems of RL

Computation of the Value Function v(s)

Computation of the Policy Function $\pi(s)$

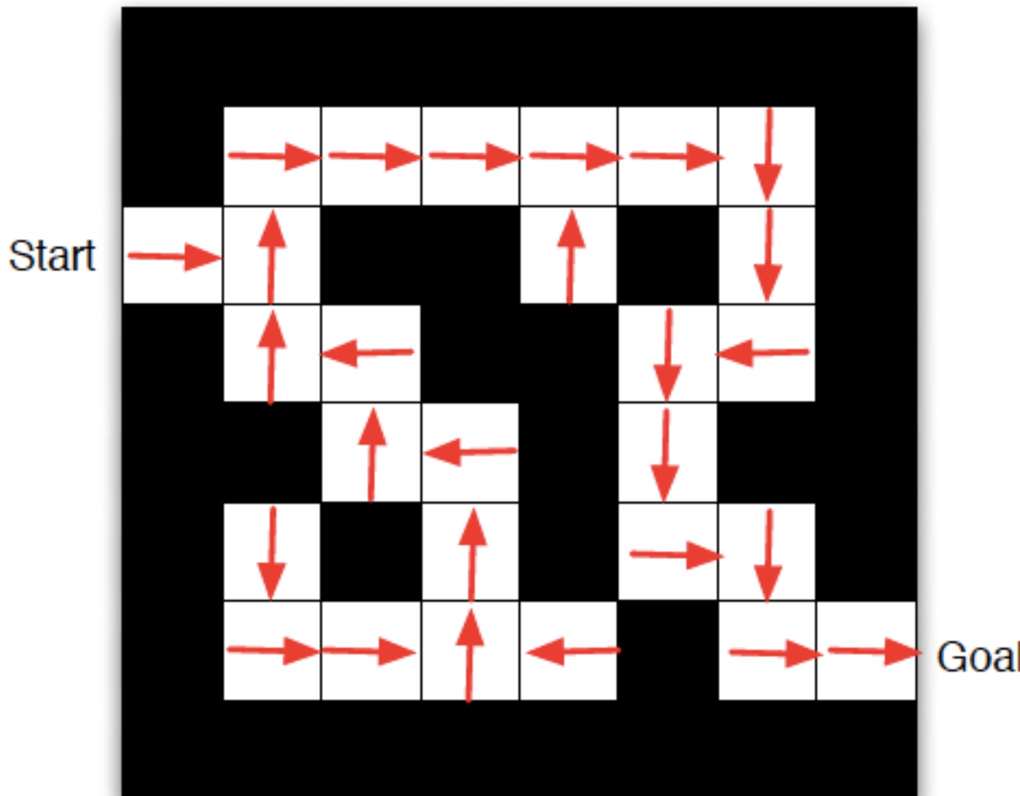
Example

Maze Example



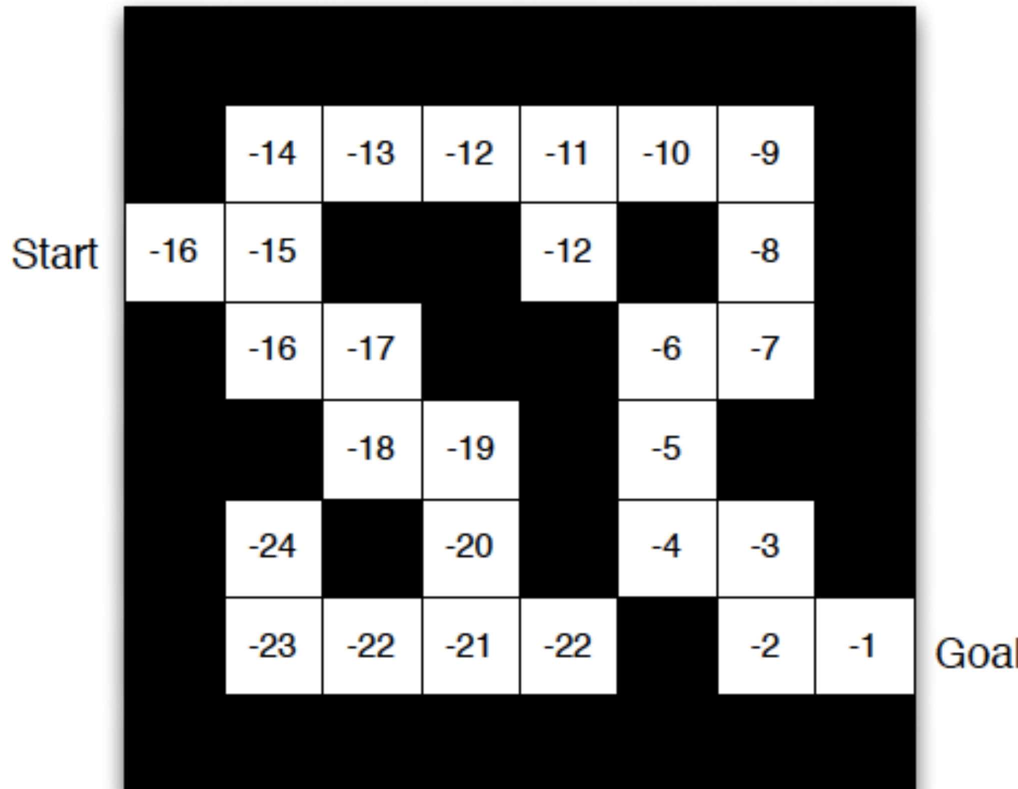
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

Maze Example: Policy



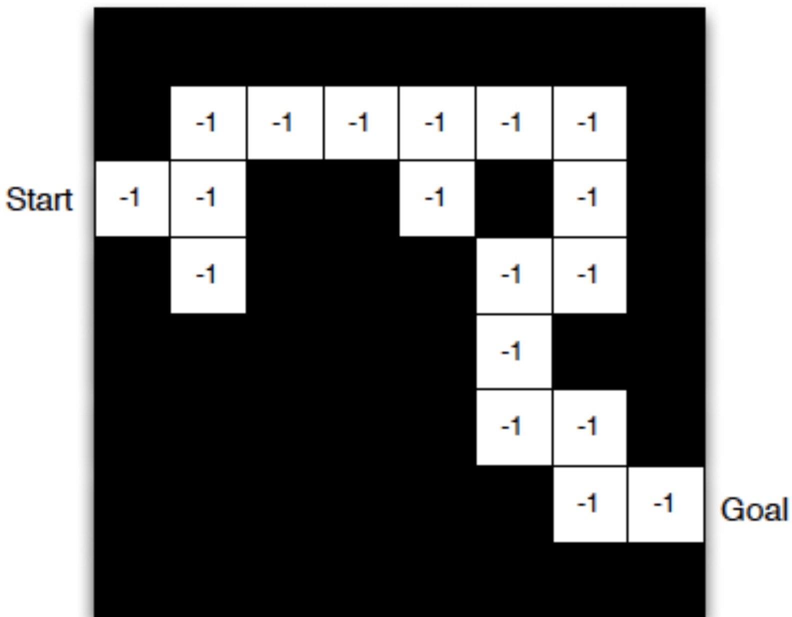
- Arrows represent policy $\pi(s)$ for each state s

Maze Example: Value Function



- Numbers represent value $v_\pi(s)$ of each state s

Maze Example: Model



- Agent may have an internal model of the environment
- Dynamics: how actions change the state
- Rewards: how much reward from each state
- The model may be imperfect

- Grid layout represents transition model $\mathcal{P}_{ss'}^a$
- Numbers represent immediate reward $\mathcal{R}_s^a$ from each state s (same for all a)

RL Agent Taxonomy

Categorizing RL Agents

- Value Based
  - No Policy (Implicit)
  - Value Function

Objective: Learn v(s)

- Policy Based
  - Policy
  - No Value Function

Objective: Learn $\pi(s)$

- Actor Critic
  - Policy
  - Value Function

Objective: Learn Both v(s) and $\pi(s)$

Categorizing RL Agents (cont)

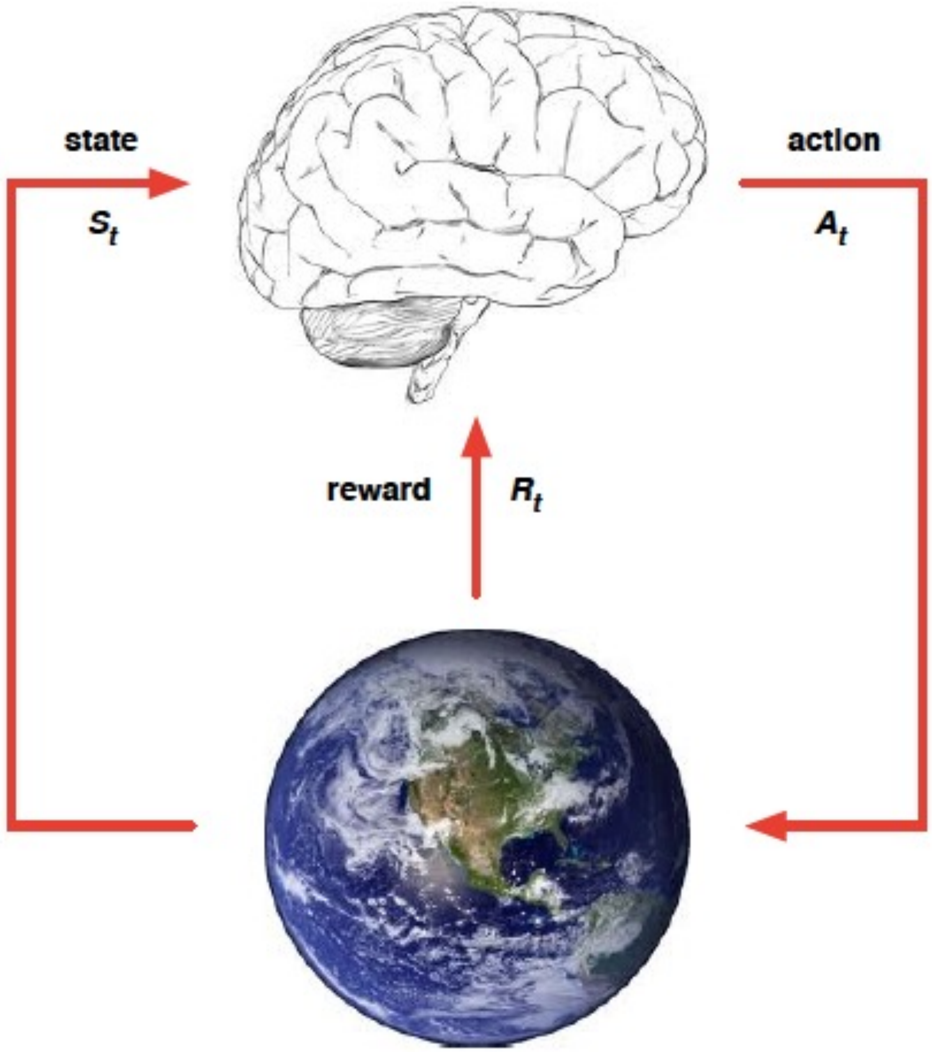
- Model Free
  - Policy and/or Value Function
  - No Model

Take Action and proceed by Trial and Error

- Model Based
  - Policy and/or Value Function
  - Model

Plan ahead before taking Action

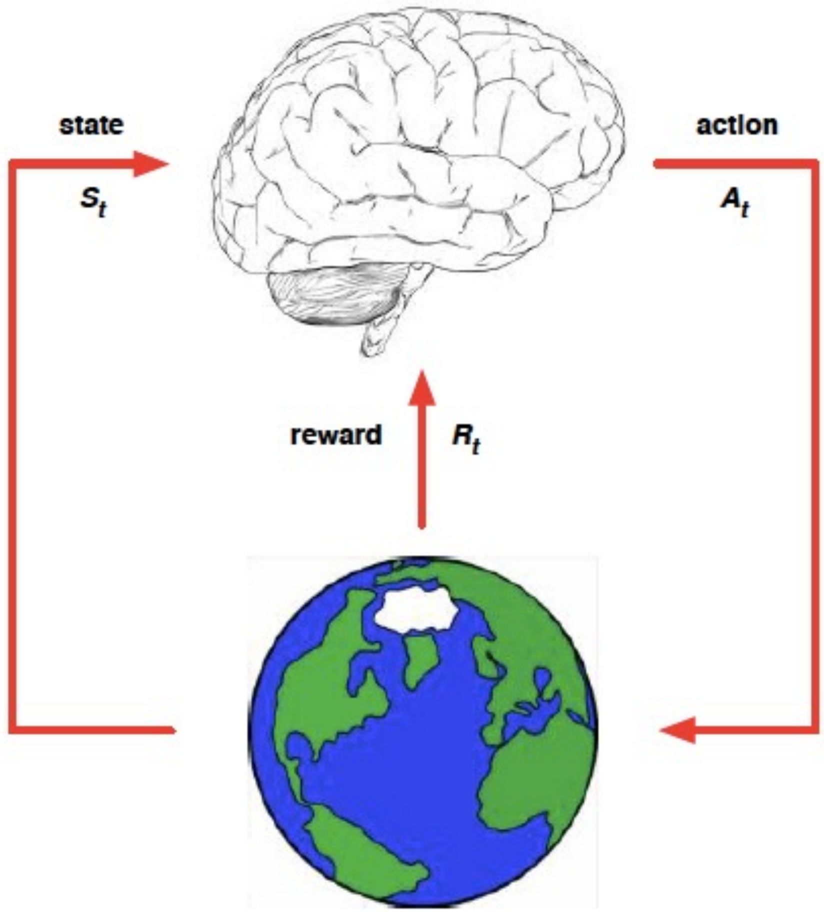
Model Free RL



The Agent does not have any visibility into how the next State and Reward are being generated by the environment

Model based RL

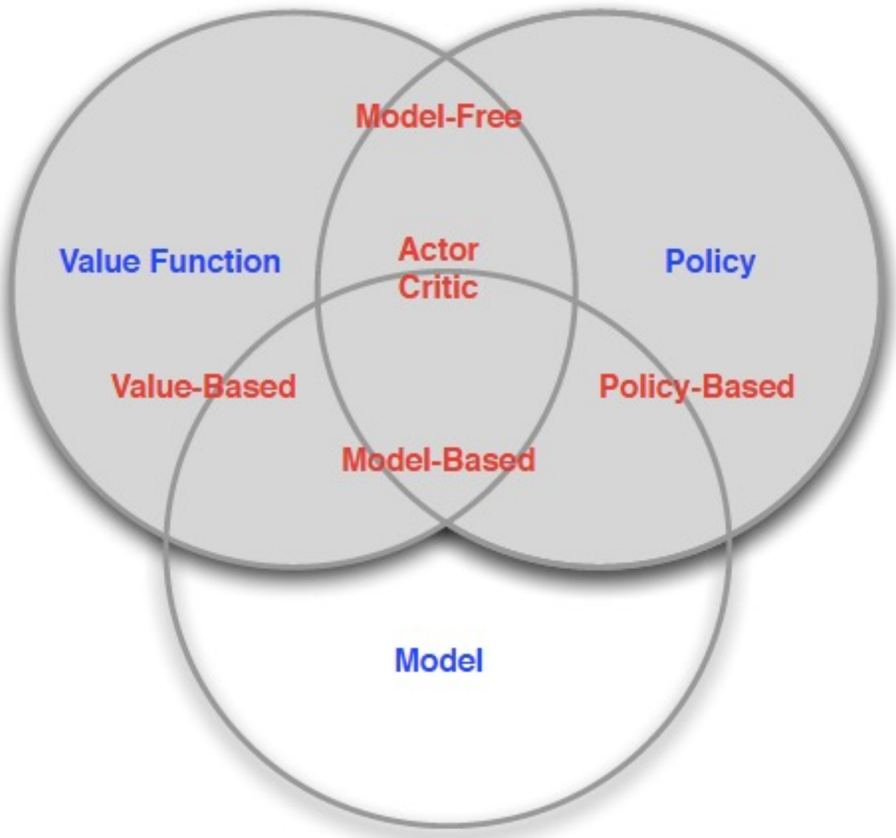
Agent



Agent's World Model

The Agent has a Model for the environment

RL Agent Taxonomy



Sub-Problems within RL

Learning and Planning

Two fundamental problems in sequential decision making

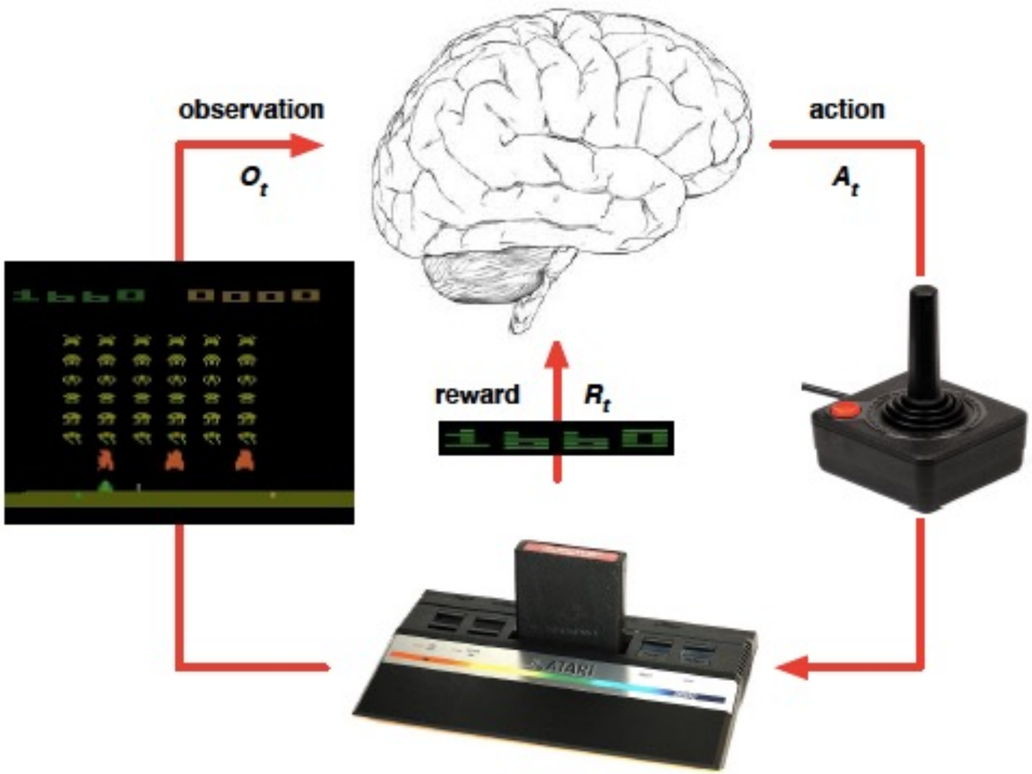
- Reinforcement Learning:
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy

Model Free

- Planning:
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy
  - a.k.a. deliberation, reasoning, introspection, pondering, thought, search

Model Based

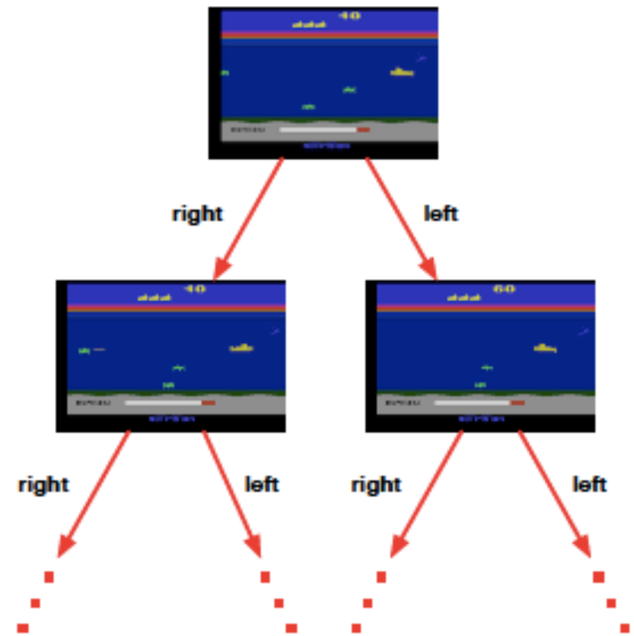
Atari Example: Reinforcement Learning



- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores

Atari Example: Planning

- Rules of the game are known
- Can query emulator
  - perfect model inside agent's brain
- If I take action *a* from state *s*:
  - what would the next state be?
  - what would the score be?
- Plan ahead to find optimal policy
  - e.g. tree search



Exploration and Exploitation

- Reinforcement learning is like trial-and-error learning
- The agent should discover a good policy
- From its experiences of the environment
- Without losing too much reward along the way

- *Exploration* finds more information about the environment
- *Exploitation* exploits known information to maximise reward
- It is usually important to explore as well as exploit

Examples

- Restaurant Selection

Exploitation	Go to your favourite restaurant
Exploration	Try a new restaurant

- Online Banner Advertisements

Exploitation	Show the most successful advert
Exploration	Show a different advert

- Oil Drilling

Exploitation	Drill at the best known location
Exploration	Drill at a new location

- Game Playing

Exploitation	Play the move you believe is best
Exploration	Play an experimental move

Prediction and Control

- Prediction: evaluate the future
    - Given a policy
- Control: optimise the future
    - Find the best policy

Example: Driving to Work Everyday

- Environment: All the roads between Home and Work, with random traffic loads
- Action: At each Intersection – Go Straight, Go Left, Go Right
- Reward: –(Time elapsed)
- State: What we see in Front (+Side and Backview mirrors)

Example: How to Find the Best Route to Work

- Algorithm 1: Trial and Error
  - Repeat N Times
    - Try out a route – i.e. choose a Policy
    - Keep track of delays while carrying out Policy
  - Choose optimal route based on delays observed while following the N Routes
- Algorithm 2: Model Based
  - Before starting commute, consult Google Maps. Run some scenarios based on the traffic.
  - Choose Route with least traffic.

Deep Reinforcement Learning

Deep Reinforcement Learning

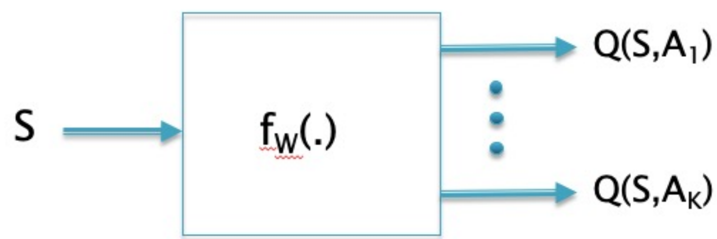
Two Types of Reinforcement Learning Algorithms:

1. Tabular Reinforcement Learning

	A1	A2	A3	A4
S1	Q(S1,A1)	Q(S1,A2)	Q(S1,A3)	Q(S1,A4)
S2	Q(S2,A1)	Q(S2,A2)	Q(S2,A3)	Q(S2,A4)
S3	Q(S3,A1)	Q(S3,A2)	Q(S3,A3)	Q(S3,A4)
S4	Q(S4,A1)	Q(S4,A2)	Q(S4,A3)	Q(S4,A4)

This approach does not scale if the number of states is very large (in the multiple millions)

2. Deep Reinforcement Learning

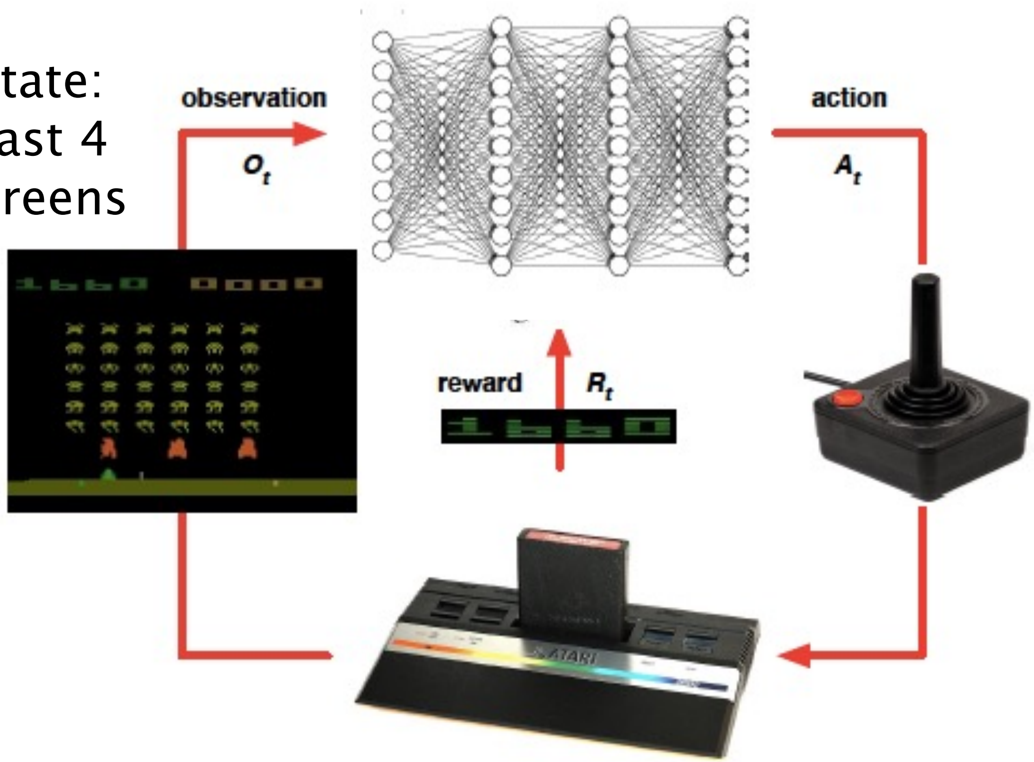


f_W is represented using a multilayer Neural Network

Deep RL : Atari Example

Agent: Player

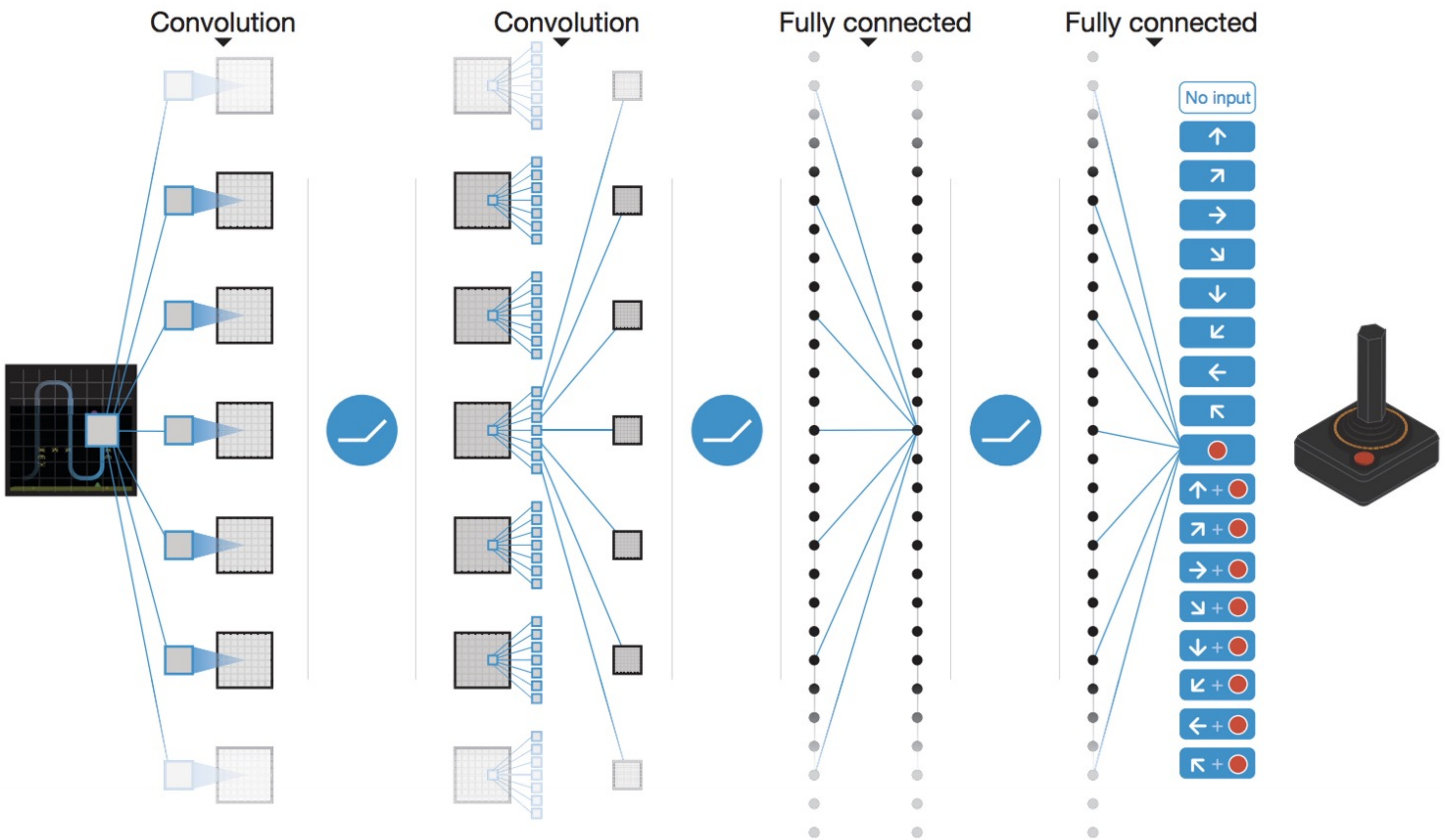
State:
Last 4
Screens



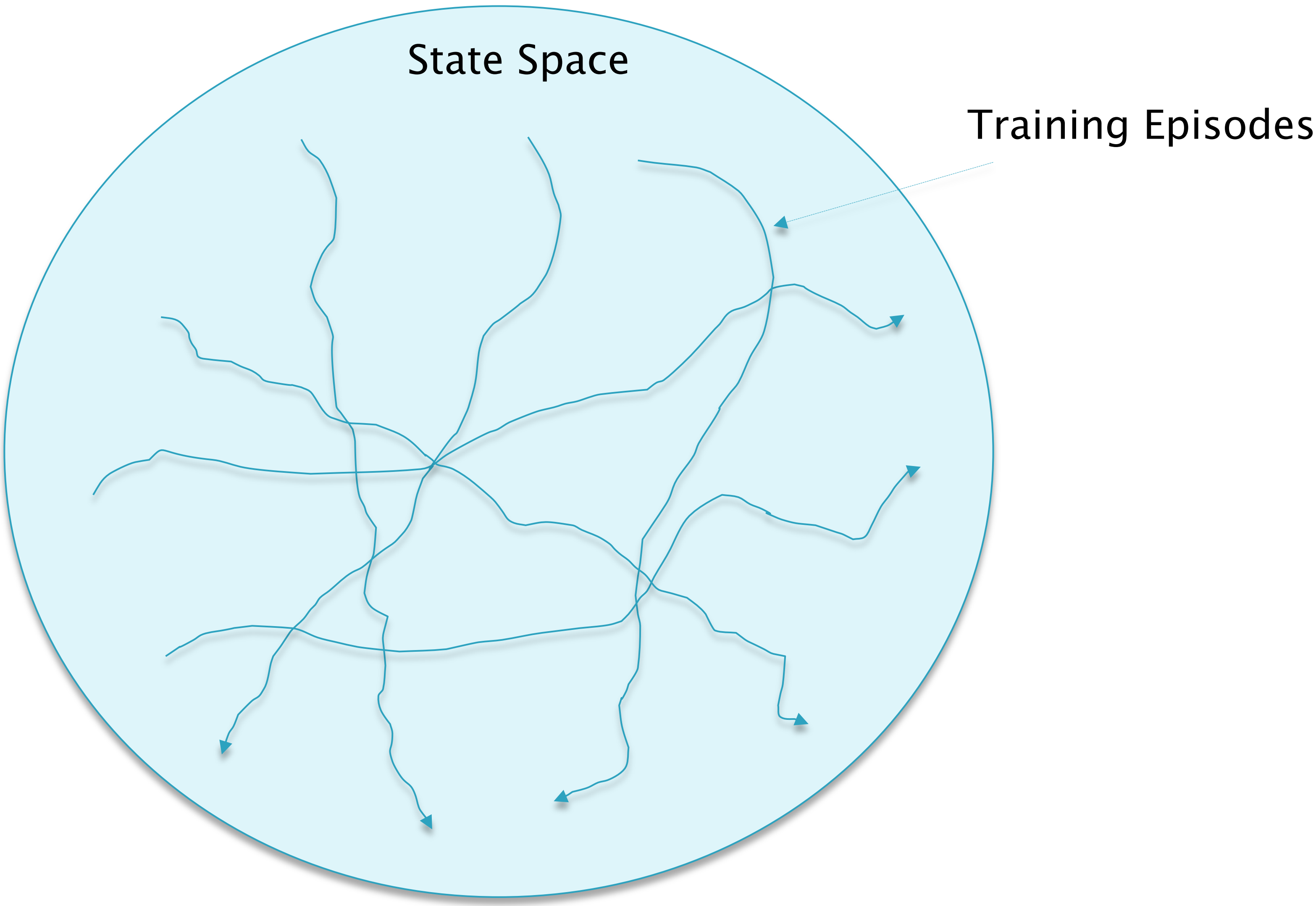
Environment: Game Software

Agent is implemented using a Neural Network

Deep Reinforcement Learning



Deep Models allows RL algorithms to solve Complex Decision Making Problems End-to-End



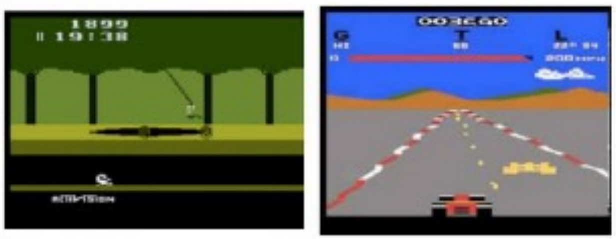
Neural Network approximates the Value Function for parts of the State space outside the sample episodes

Deep Reinforcement Learning

- Deep = can process complex sensory input
  - ...and also compute really complex functions
- Reinforcement learning = can choose complex actions

Deep Reinforcement Learning: Can Solve Complex Decision Making Problems with Complex Sensory Input

Recent Successes of Deep RL



Atari games:

Q-learning:

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, et al. "Playing Atari with Deep Reinforcement Learning". (2013).

Policy gradients:

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust Region Policy Optimization". (2015).

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, et al. "Asynchronous methods for deep reinforcement learning". (2016).

Real-world robots:

Guided policy search:

S. Levine*, C. Finn*, T. Darrell, P. Abbeel. "End-to-end training of deep visuomotor policies". (2015).

Q-learning:

S. Gu*, E. Holly*, T. Lillicrap, S. Levine. "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates". (2016).

Beating Go champions:

Supervised learning + policy gradients + value functions + Monte Carlo tree search:

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, et al. "Mastering the game of Go with deep neural networks and tree search". Nature (2016).

Lecture Schedule

- **Lecture 1 – Introduction to Reinforcement Learning**: Introduction to Reinforcement Learning and discussion of important applications, An historical overview of the development of this topic.
- **Lecture 2 – Markov Decision Processes**: Markov Processes, Markov Reward Process, Value Function, Markov Decision Processes, Policies, Bellman Expectation Equation, Optimal Value Function, Optimal Policies, Bellman Optimality Equation.
- **Lecture 3 – Planning by Dynamic Programming**: Estimating the Value Function of a known MDP by Dynamic Programming, Policy Evaluation, Policy Iteration, Value Iteration.
- **Lecture 4 – Model Free Prediction**: Estimating the Value Function of an unknown MDP, Monte Carlo (MC) based Policy Evaluation, Temporal Difference (TD) Learning, Comparison of MC and TD Methods.
- **Lecture 5 – Model Free Control**: Optimizing the Value Function of an Unknown MDP, Epsilon Greedy Policies, On Policy Monte Carlo Control, On Policy Temporal Difference Control, SARSA Control, Off Policy Learning, Q-Learning.

- **Lecture 6 – Overview of Deep Learning Neural Networks**: Supervised Learning, Function Approximations using Deep Learning, Training Algorithms, Convolutional and Recurrent Neural Networks
- **Lecture 7 – Value Function Approximation using Deep Learning**: Large Scale Reinforcement Learning, Types of Value Function Approximations (VFA), VFA using Deep Learning Networks, Monte Carlo based VFA, Temporal Difference based VFA, Deep Q Networks (DQN), Advanced DQN Algorithms.
- **Lectures 8 – Policy Gradient Methods**: Policy based Reinforcement Learning, Policy Optimization, Policy Gradient, Monte Carlo based Policy Gradient (REINFORCE), Actor-Critic Algorithms.
- **Lectures 9 – Integrating Learning and Planning**: Model based Reinforcement Learning, Learning Models from experience, Planning with a Model, Integrated Learning and Planning, Dyna-Q Algorithm, Monte Carlo Tree Search (MCTS) Algorithm, AlphaGo Zero Algorithm
- **Lectures 10 – RLHF**: Reinforcement Learning based on Human Feedback, Large Language Models, Reward Models, Proximal Policy Optimization (PPO) Algorithm

Further Reading

Sutton and Barto:

- Chapter 1
- Chapter 3: Sections 3.1 – 3.4