Markov Decision Processes

CMPUT 366: Intelligent Systems

S&B §3.0-3.4

Lecture Outline

1. Recap & Logistics
2. Markov Decision Processes
3. Returns & Episodes

Logistics

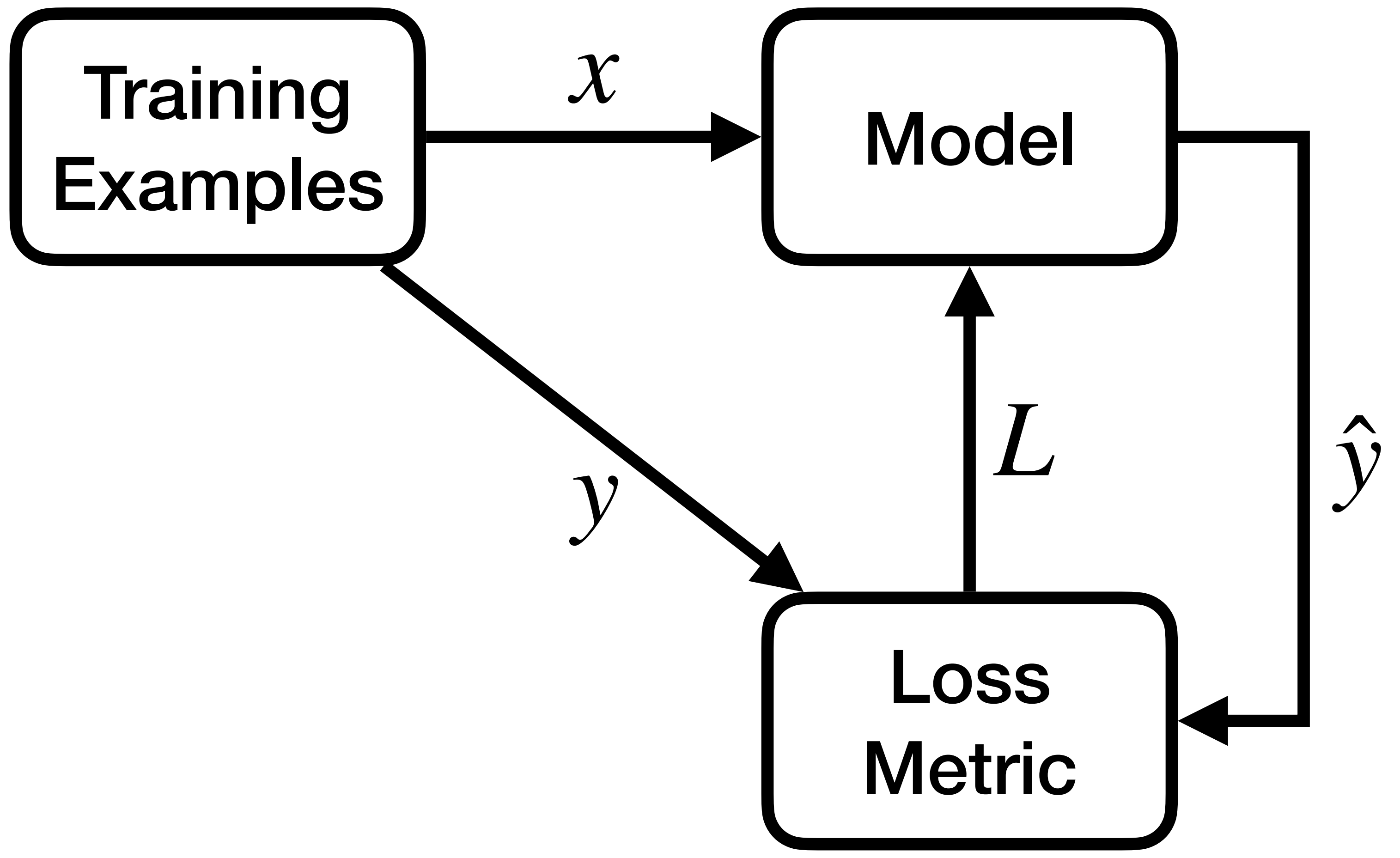
- **Midterm** is being marked
    - Spot checks: You will be contacted next week if selected
- **Assignment 3** is available
    - Due: Fri, March 25 at 11:59pm Mountain time
    - Stub code uses Tensorflow **version 2.8.0**
    - OS X installation:
      `python3 -m pip install tensorflow`

Recap: Deep Learning

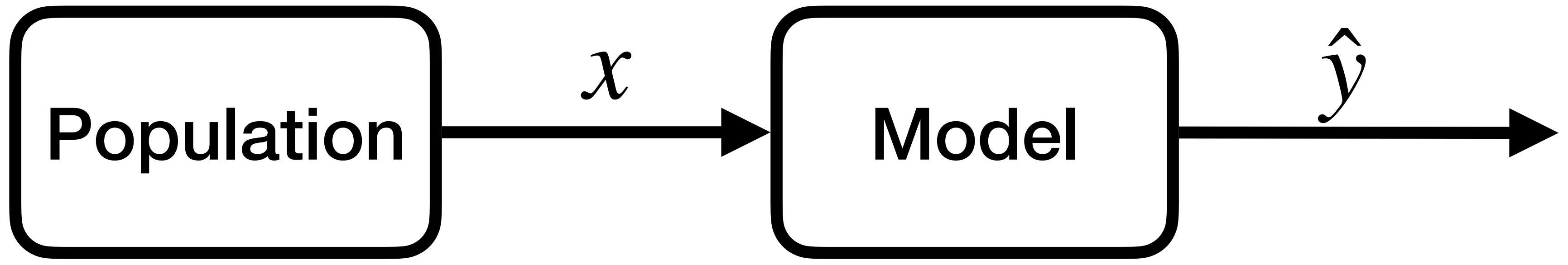
- **Feedforward neural networks** are **extremely flexible** parametric models that can be trained by gradient descent
- **Convolutional neural networks** add **pooling** and **convolution** operations
  - Vastly more efficient to train on **vision** tasks, due to **fewer parameters** and domain-appropriate **invariances**
- **Recurrent neural networks** process **elements of a sequence** one at a time, usually while maintaining **state**
  - Same set of **weights** applied to each element

Recap: Supervised Learning

Neural networks are generally used to solve **supervised learning** tasks: Selecting a **hypothesis** $h : X \to Y$ that maps from **input** features to **target** features.



Training time



Test time

Example: CanBot

- CanBot's job is to find and recycle empty cans
- At any given time, its battery charge is either **high** or **low**
- It can do three actions: **search** for cans, **wait**, or **recharge**
- *Goal:* Find cans efficiently without running out of battery charge

Questions:

1. Is this an instance of a **supervised learning** problem?
2. Is this an instance of a **search** problem?

Reinforcement Learning

In a **reinforcement learning** task, an agent learns how to **act** based on feedback from the **environment**.

- The agent's actions may change the environment
- The "right answer" is not known
- The task may be either **episodic** or **continuing**
- The agent makes decisions **online**: determines how to act while interacting with the environment

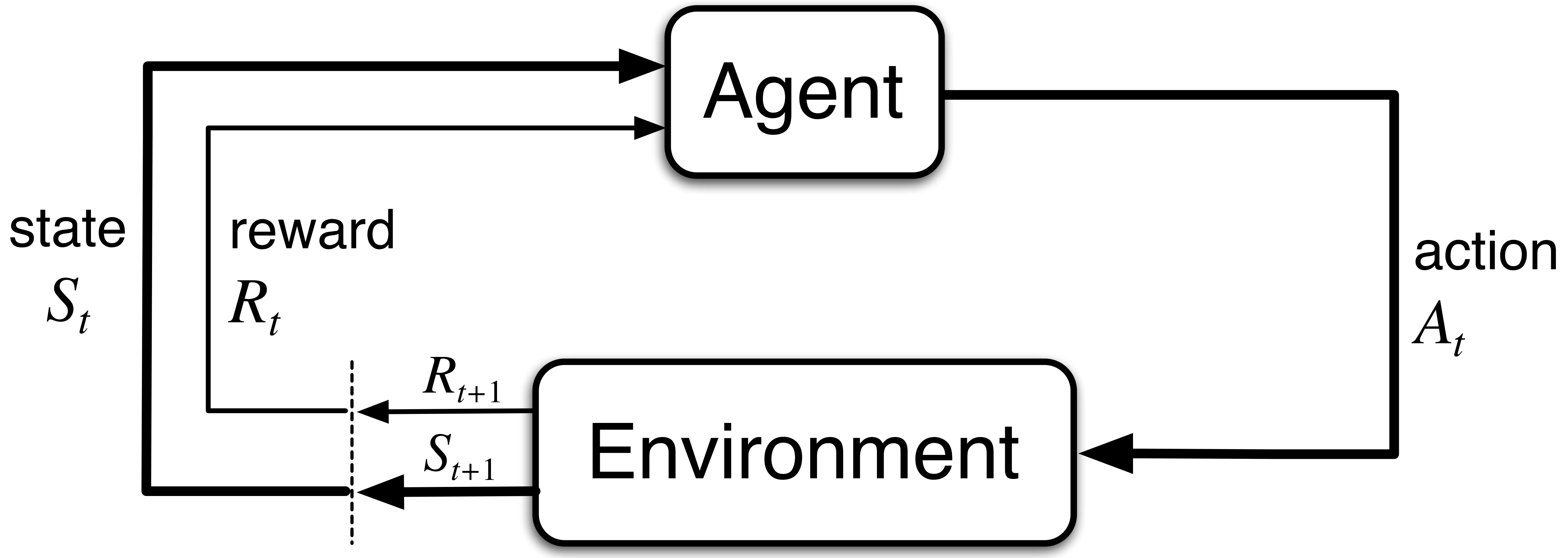
Interacting with the Environment

At each time $t = 1,2,3,\ldots$

1. Agent receives input denoting **current state** S_t

2. Agent chooses **action** A_t

3. Next time step, agent receives **reward** R_{t+1} and **new state** S_{t+1}, chosen according to a distribution $p(s', r \mid s, a)$



This interaction between agent and environment produces a **trajectory**:
$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$

Markov Decision Process

Definition:

A **Markov decision process** is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$, where

- $\mathcal{S}$ is a set of **states**,
- $\mathcal{A}$ is a set of **actions**,
- $\mathcal{R} \in \mathbb{R}$ is a set of **rewards**,
- $p(s', r \mid s, a) \in [0,1]$ defines the **dynamics** of the process, and
- the probabilities from p **completely** characterize the environment's dynamics

Dynamics

The four-argument dynamics function returns the probability of every **state transition**:

$$p(s', r \mid s, a) \doteq \Pr(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a)$$

It is often convenient to use **shorthand notation** rather than the full four-argument dynamics function:

$$p(s' \mid s, a) \doteq \Pr(S_t = s' \mid S_{t-1} = s, A_{t-1} = a) = \sum_{r \in \mathcal{R}} p(s', r \mid s, a)$$

$$r(s, a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a)$$

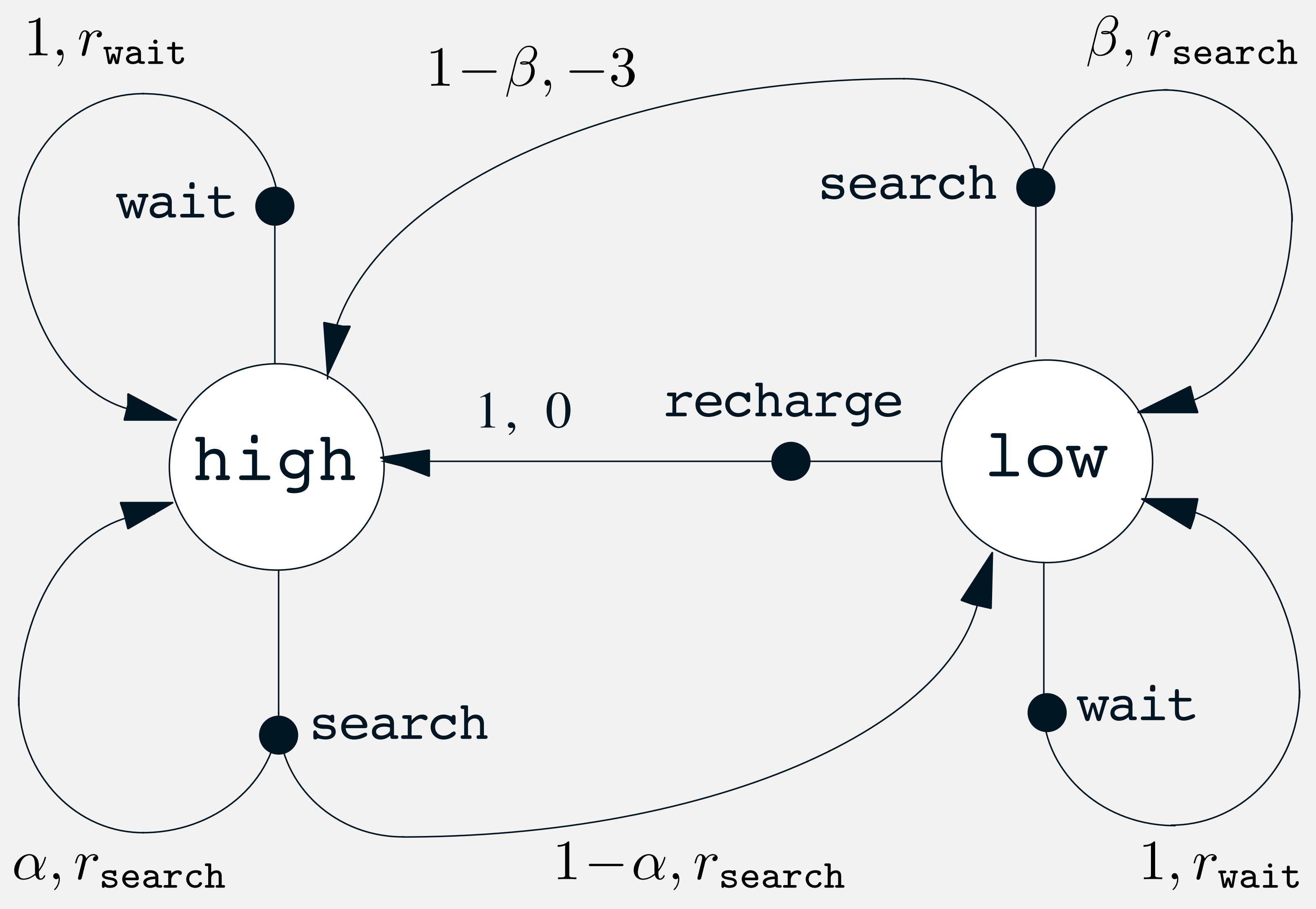
$$r(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}$$

CanBot as a Reinforcement Learning Agent

Question: How can we represent CanBot as a **reinforcement learning** agent?

- Need to define **states**, **actions**, **rewards**, and **dynamics**

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1 - \alpha$	$r_{\texttt{search}}$
low	search	high	$1 - \beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	-



(Image: Sutton & Barto, 2018)

Reward Hypothesis

Definition: *Reward hypothesis*
An agent's goals and purposes can be entirely represented as the maximization of the **expected value** of the **cumulative sum** of a **scalar signal**.

Returns for Episodic Tasks

Question: What does it *mean* to maximize the expected value of the cumulative sum of rewards?

Definition: A task is **episodic** if it ends after some **finite number** T of time steps in a special **terminal state** S_T.

Definition: The **return** G_t after time t is the sum of rewards received after time t: $G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$.

Answer: The return G_t is a **random variable**. In an episodic task, we want to maximize its **expected value** $\mathbb{E}[G_t]$.

Returns for Continuing Tasks

Definition: A task is **continuing** if it does not end (i.e., $T = \infty$).

- In a continuing task, we can't just maximize the sum of rewards (**why?**)
- Instead, we maximize the **discounted return**:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$
$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$\gamma \leq 1$ is the **discount factor**

- Returns are **recursively** related to each other:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$
$$= R_{t+1} + \gamma G_{t+1}$$

Summary

- **Supervised learning models** are trained **offline** using **labelled training examples**, and then make **predictions**
- **Reinforcement learning agents** choose their **actions online**, and update their behaviour based on **rewards** from the **environment**
- We can formally represent reinforcement learning environments using **Markov decision processes**, for both **episodic** and **continuing** tasks
- Reinforcement learning agents maximize **expected returns**