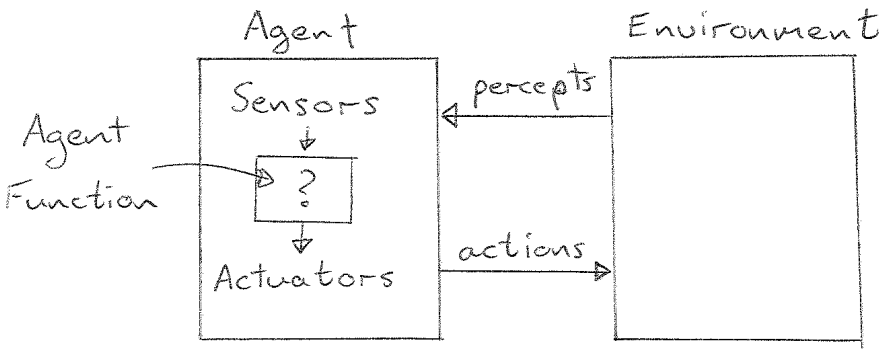
COMS 4030A

Adaptive Computation and Machine Learning

12. Reinforcement Learning

12.1. Intelligent Agents.

A basic schematic diagram for an intelligent **agent** acting within an environment is as follows:



The agent has sensors with which it receives information, called **percepts**, from its environment. The information received is fed into the **agent function** which makes a decision on an **action** to take. The decision is fed to the actuators which perform the action. This action has effects on the agent and its environment, and the new information is received by the agent's sensors, which cause the agent to take a new action, and so on. Usually the agent has some **goal** which guides the choice of actions it takes.

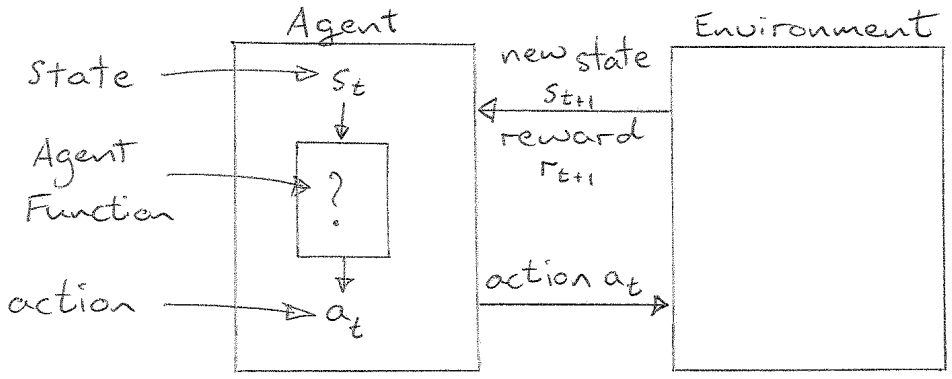
Examples: Consider a Vacuum-Cleaner Robot, which operates in a closed room - the environment. The sensors on the robot may include some sort of positioning sensor, cameras, object detector, dirt sensor, etc. The information received from these sensors cause the robot to take an action, which may be to turn, move forward, suck up dirt, etc. The actuators are then the mechanical parts that cause it to move, turn, and so on. A goal for this agent would be to ensure there is no dirt in the room.

Consider a chess-playing computer. The environment for this agent is a chess board, and the information it receives is the positions of the pieces on the board and moves made by the opponent. The agent function (using lots of computation) decides on the move the agent should make. The goal for this agent is to win the game.

Consider a self–driving car. Its environment could be all the roads in a city, or a country. Its sensors could include speedometer, cameras, GPS, internet connection, which give it information such as position, direction and speed of travel, nearby vehicles. The agent function must make decisions such as change speed, change direction, stop, etc., with the goal of getting to some destination without a collision.

12.2. **Reward functions.**

Reinforcement learning is an approach to intelligent agents that makes use of a **reward function** to allow the agent to **learn** a good agent function from experience. The basic idea is that whenever an agent takes a good action, one which helps it towards its goal, it gets a positive reward (i.e., a positive number) and for any action that does not help it towards its goal, it receives a negative reward, or zero reward. In this way the agent should be able to learn to take actions that maximize its long-term rewards. A basic schematic diagram is as follows.



The information that the agent receives from its sensors tells it the **state** it is in at time-step t, say s_t (s_t is a list of all the information it has about its environment). The agent function uses the current state s_t to decide on an action a_t to take at time-step t. This action causes changes to the agent's state, determined by the environment, and the new state s_{t+1} at time-step $t+1$ is received by the agent. At the same time, the environment returns a **reward** r_{t+1} to the agent (where $r_{t+1} \in \mathbb{R}$). The reward obtained is determined by a **reward function**, which must be chosen in advance in such a way as to help the agent achieve its goal.

Examples: In the Vacuum-cleaner robot example, a reward function could be 1 for each time it collects dust, and 0 otherwise (or -1 otherwise). The long-term goal is to collect as much dust as possible.

In the chess-playing computer example, the reward function could be 100 for any action that wins the game, -100 for any action that loses the game, 50 for a draw, say, and 0 otherwise. Remember that the agent is trying to maximize its long-term rewards, so a sequence of actions that return 0 for a long time, but eventually return 100 is better than a sequence of actions that return 0 for a long time and then returns -100.

In the self-driving car example, a reward function could be 100 for arriving at its destination, and -1000 for a collision, and -1 otherwise. (It would be better to train the agent in a simulated environment first.)

12.3. **The Markov Property.**

One important assumption that is usually made in reinforcement learning is the Markov property. This property states that if an agent is in state s_t and takes action a_t, then the resulting state s_{t+1} depends only on the state s_t and the action a_t. That is, the states and actions prior to s_t and a_t, i.e., $\dots s_{t-2}$, s_{t-1} and actions $\dots a_{t-2}, a_{t-1}$ no longer influence the new state. Similarly, the reward r_t only depends on s_t, a_t and s_{t+1}. The Markov property is used in order to reduce the size of the state space. Without this property the resulting state after an action could be dependent on the whole sequence of previous states and actions.

An example in which the Markov-property **does not** hold is the following. Consider a self-driving car, in which the only information in your state s_t is the GPS information. If the car takes an action a_t to turn right, then the resulting state s_{t+1} does not only depend on s_t and a_t, but on the previous actions as well. For example, if the action a_{t-1} was to stop, then turning right has a different result to if a_{t-1} was to go faster. In this example, the Markov property is true if we include in the state information also the velocity of the car, since the resulting state depends only on the position and velocity and action taken.

12.4. **Markov Decision Processes.**

The mathematical framework for reinforcement learning is a **Markov Decision Process**, or an **MDP**. An MDP is comprised of the following:

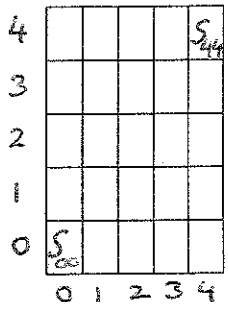
• A set S of **states**.

• A set A of **actions** (all actions may all be available in all states, or it may be that in some states only a subset of actions is available).

• A **state transition function** $P : S \times A \times S \to [0, 1]$, such that for every state $s \in S$ and every action $a \in A$, we have that $\sum_{s' \in S} P(s, a, s') = 1$.
This function is intepreted as follows: for state $s \in S$, action $a \in A$ and state $s' \in S$, $P(s, a, s')$ is a number in $[0, 1]$ that gives the probability that the agent ends up in state s', given that it is in state s and takes action a.

• A **reward function** $R : S \times A \times S \to \mathbb{R}$.
This function is intepreted as follows: if an agent is in state $s \in S$, takes action $a \in A$ and ends up in state $s' \in S$, then $R(s, a, s')$ is the reward it receives.

Examples: The following is an example of a **grid-world** MDP:



Each cell in the grid is a state. Let state s_{ij} be the cell in position i horizontally and j vertically.
The sets of states is then $S = \{s_{ij} \mid 0 \leq i \leq 4,\ 0 \leq j \leq 4\}$.
The set of actions is $A = \{\ell, u, r, d\}$ corresponding to left, up, right, down.
The transition rules are simply that the agent moves 1 cell left if action ℓ is take, 1 cell up if action u is taken, 1 cell right if action r is taken and 1 cell down if d is taken.
If an action would take the agent off the grid, then he stays in the current state he is in.

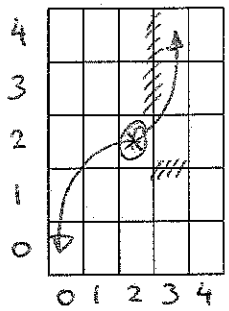
The transition function P can be given as follows:
$P(s_{00}, u, s_{01}) = 1$, $P(s_{00}, u, s_{ij}) = 0$ for any s_{ij} other than s_{01}.
$P(s_{00}, r, s_{10}) = 1$, $P(s_{00}, r, s_{ij}) = 0$ for any s_{ij} other than s_{10}.
$P(s_{00}, \ell, s_{00}) = 1$, $P(s_{00}, \ell, s_{ij}) = 0$ for any s_{ij} other than s_{00}.
$P(s_{00}, d, s_{00}) = 1$, $P(s_{00}, d, s_{ij}) = 0$ for any s_{ij} other than s_{00}.
$\vdots$
$P(s_{12}, r, s_{22}) = 1$, $P(s_{12}, r, s_{ij}) = 0$ for any s_{ij} other than s_{22}.
$\vdots$
$P(s_{34}, d, s_{33}) = 1$, $P(s_{34}, d, s_{ij}) = 0$ for any s_{ij} other than s_{33}.
$\vdots$

Suppose that the goal of the agent is to start at s_{00} and get to s_{44}.
Then a reward function could be the following:
Any action that gets the agent to state s_{44} gets a reward of 100, i.e., $R(s_{ij}, a, s_{44}) = 100$.
Any other action gets a reward of 0, i.e., $R(s_{ij}, a, s_{k\ell}) = 0$ if $s_{k\ell} \neq s_{44}$.

With this reward function, any path from s_{00} to s_{44} gets a total reward of 100. Thus, a suboptimal path that visits every cell is as good as a shortest path. The reward function can be improved by specifying that any action other than an action that gets you to s_{44} gets a reward of -1, i.e., $R(s_{ij}, a, s_{k\ell}) = -1$ if $s_{k\ell} \neq s_{44}$. In this way, shorter paths are encouraged. An optimal path from s_{00} to s_{44} gets a total of $-7 + 100 = 93$ by the time it reaches s_{44}.

Another change we can make is to discourage the agent from bashing into walls. For any action that would take the agent off the grid (but leaves him where he is) he gets a reward of -5. So, for example, $R(s_{00}, \ell, s_{00}) = -5$, $R(s_{10}, d, s_{10}) = -5$, etc.
With this reward function the agent should learn faster that bashing into walls is bad.

To make the gridworld more interesting we can add some features. For example, we can add some barriers between cells, that a agent cannot pass through, as in the diagram below.
(Alternatively, we could make them pseudo-barriers that have a cost of -3 to move through.)



To introduce some randomness into the gridworld MDP, we can make some cells random-cells that transport you randomly to other cells in the grid. For example, let's say that if the agent tries to move into cell s_{22}, it gets transported either to s_{34} with probability 0.7, or to s_{00} with probability 0.3. The transition probability function would be defined as follows:
$P(s_{12}, r, s_{34}) = 0.7$, $P(s_{12}, r, s_{00}) = 0.3$ and $P(s_{12}, r, s_{ij}) = 0$ for any other s_{ij}.
$P(s_{21}, u, s_{34}) = 0.7$, $P(s_{21}, u, s_{00}) = 0.3$ and $P(s_{21}, u, s_{ij}) = 0$ for any other s_{ij}.
$P(s_{32}, \ell, s_{34}) = 0.7$, $P(s_{32}, \ell, s_{00}) = 0.3$ and $P(s_{32}, \ell, s_{ij}) = 0$ for any other s_{ij}.

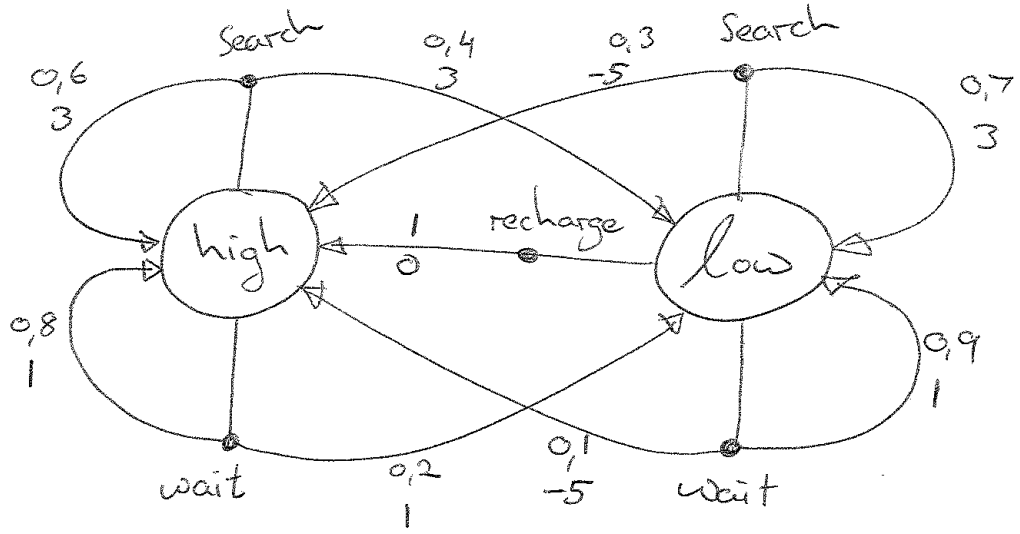
$P(s_{23}, d, s_{34}) = 0.7$, $P(s_{23}, d, s_{00}) = 0.3$ and $P(s_{23}, d, s_{ij}) = 0$ for any other s_{ij}.
The reward for each such move is -1.

Each combination of states and actions that takes the agent from s_{00} to s_{44} is called an **episode**, and the above gridworld MDP is called **episodic**.

The main question we are interested in is: What is the optimal path from s_{00} to s_{44}? Here, 'optimal' means what is the greatest sum of rewards we can expect to collect in one episode from s_{00} to s_{44}. We will look at such questions below.

First, another example:

Consider a recycling robot that wanders the streets of Tokyo to find aluminium cans to recycle. For simplicity, suppose that the only state information we have is the battery power of the robot, which can be either 'high' or 'low', so $S = \{\text{high, low}\}$. The set of actions is $A = \{\text{recharge, search, wait}\}$, but the action 'recharge' cannot be taken in state 'high'.
The following diagram gives the transition probability function and the reward function for this robot (i.e., the MDP).



For example, if the agent is in state 'high' and chooses action 'search', there is a 0.6 probability that the agent will end up in state 'high' again and a 0.4 probability that it will end up in state 'low', i.e., $P(\text{high, search, high}) = 0.6$ and $P(\text{high, search, low}) = 0.4$. The reward in both cases is 3 (representing the expected number of cans it will collect).

If the agent is in state 'high' and chooses action 'wait', there is a 0.8 probability that the agent will end up in state 'high' again and a 0.2 probability that it will end up in state 'low', i.e., $P(\text{high, wait, high}) = 0.8$ and $P(\text{high, wait, low}) = 0.2$. The reward in both cases is 1 since it is expected to collect fewer cans by simply waiting around.

If the agent is in state 'low' and chooses action 'search', there is a 0.7 probability that the agent will end up in state 'low' again, i.e., $P(\text{low, search, low}) = 0.7$ and a probability of 0.3 that it will end up in state 'high', i.e., $P(\text{low, search, high}) = 0.3$. This occurs if the robot's battery runs flat and someone must be despatched from head-office to collect the robot and transport it back to the recharge station. Since this is an undesirable event, it gets a big negative reward, say -5.

The goal for this robot is to collect as many cans as it can. There is no specific destination state for the agent and no time-limit, so we say this is a **continuing task**.

12.5. **Policies and Returns.**

Given a Markov Decision Process (MDP), a **policy** is any function $\pi : S \to A$.

That is, a policy is a function that, when given any state, returns an action to take in that state. (Policies that are probabilistic are also allowed, that is, policies that can take random actions according to some probability distributive.)
The objective is find an optimal policy.

Given an MDP and a policy π, suppose the agent starts in state s_0 and takes actions determined by the policy π. Say $\pi(s_0) = a_0$; then the agent takes action a_0, which results in a reward r_1 and a new state s_1, from which the agent takes a new action according to π, and continues in this way. This will generate a sequence as follows:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, \ldots, s_{T-1}, a_{T-1}, r_T, s_T.$$

The action a_0 is determined by π, namely $a_0 = \pi(s_0)$ and, similarly, $a_1 = \pi(s_1)$, $a_2 = \pi(s_2)$, etc.

In an episodic MDP, this sequence should end when a goal state is reached by the agent.
The **return** for this episode is defined as the sum of all rewards obtained in the episode, i.e.,

$$R^\pi(s_0) = r_1 + r_2 + \cdots + r_T = \sum_{i=1}^{T} r_i.$$

In a continuing MDP, this sequence could continue indefinitely, which gives an infinite sum. To deal with this, a factor γ is included into the rewards, giving the following **discounted return**:

$$R^\pi(s_0) = r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots + \gamma^{i-1} r_i + \cdots = \sum_{i=1}^{\infty} \gamma^{i-1} r_i.$$

The number γ is a fixed value with $0 \leq \gamma \leq 1$, called the **discount factor**. If $\gamma < 1$ then the infinite series above converges. A common value for γ is 0.9; then $\gamma^2 = 0.81$, $\gamma^3 = 0.72$, $\gamma^4 = 0.63$, etc.

The discounted return counts immediate rewards as more important and future rewards diminish. A discounted return can be used in the episodic case as well if we want to weight immediate rewards more highly.

12.6. **Value functions.**

Given an MDP, a policy π and any state s, the **value** of s with respect to π, denoted by $V^\pi(s)$, is the *expected* return if an agent starts in state s and an episode is generated by following the policy π, that is

$$V^\pi(s) = \mathbb{E}\left(R^\pi(s)\right).$$

If the MDP has no random behaviour, i.e., it is **deterministic**, then every episode starting in state s will generate the same return, and this is $V^\pi(s)$. However, if the MDP has some random behaviour, then, starting in s, there may be a number of different episodes that occur and each one may give a different return. In this case, $V^\pi(s)$ is the expected return over all possible episodes, i.e., the average return based on the probability of the episode occurring.

Consider the grid world above. Let π be the policy that has $\pi(s_{00}) = r$, $\pi(s_{10}) = r$, $\pi(s_{20}) = r$, $\pi(s_{30}) = r$, $\pi(s_{40}) = u$, $\pi(s_{41}) = u$, $\pi(s_{42}) = u$, $\pi(s_{43}) = u$. (In fact, π must be defined for all states, but we just need the above ones for now.) Then every episode starting from s_{00} will be the same, namely:

$$s_{00}, r, -1, s_{10}, r, -1, s_{20}, r, -1, s_{30}, r, -1, s_{40}, u, -1, s_{41}, u, -1, s_{42}, u, -1, s_{43}, u, 100, s_{44}.$$

The return for this episode is 93. Thus, the value of this state is $V^\pi(s_{00}) = 93$.

Now, let π be the following policy that takes the agent through the random cell: $\pi(s_{00}) = r$, $\pi(s_{10}) = u$, $\pi(s_{11}) = r$, $\pi(s_{21}) = u$, $\pi(s_{34}) = r$. (Again the policy should be defined for all states, but we just need these for now.) There are a number of different episodes that could

happen, depending on what randomly happens when the agent moves in the direction of s_{22}. For example,

$$s_{00}, r, -1, s_{10}, u, -1, s_{11}, r, -1, s_{21}, u, -1, s_{34}, r, 100, s_{44}.$$

The return for this episode is 96. Another possible episode is:

$$s_{00}, r, -1, s_{10}, u, -1, s_{11}, r, -1, s_{21}, u, -1, s_{00}, r, -1, s_{10}, u, -1, s_{11}, r, -1, s_{21}, u, -1, s_{34}, r, 100, s_{44}.$$

The return for this episode is 92. In fact there are infinitely many different episodes that could occur depending on how many times the agent is sent back to s_{00}.
The value of s_{00} under this policy, i.e., $V^{\pi}(s_{00})$, is the expected return over all the episodes. The expected return is essentially an average of the returns over all episodes, weighted by how likely it is for each episode to occur.

12.7. **Optimal Policies.**

Given an MDP, a policy π^* is called an **optimal policy** if $V^{\pi^*}(s) \geq V^{\pi}(s)$ for all other policies π and all states s. That is, the value of following policy π^* from any state s is better or equal to the value of following any other policy.

Consider an MDP for which the state transition function P and the reward function R are known. The optimal policy π^* for this MDP can be obtained by first computing the value function V^{π^*} using **Bellman's Optimality Equation**, as follows. For every $s \in S$,

$$V^{\pi^*}(s) = \max_{a \in A} \sum_{s' \in S} P(s, a, s') \left(R(s, a, s') + \gamma V^{\pi^*}(s') \right).$$

Note that the above equation actually represents a whole system of equations – one for each $s \in S$. The unknowns in the system are the values of $V^{\pi^*}(s)$ for each $s \in S$.

If we can solve these equations, i.e., find $V^{\pi^*}(s)$ for each $s \in S$, then we can determine the optimal policy π^* as follows. For every $s \in S$,

$$\pi^*(s) = \operatorname*{argmax}_{a \in A} \sum_{s' \in S} P(s, a, s') \left(R(s, a, s') + \gamma V^{\pi^*}(s') \right).$$

The above formula states that if the agent is in state s, then the optimal action to take is the action a that maximizes the summation (i.e., the argmax).