Reinforcement Learning

Review: Types of Learning

- There are three broad types of learning:
- Supervised learning
  - Learner looks for patterns in inputs. Teacher tells learner the “right”or “wrong” answer.
- Unsupervised learning
  - Learner looks for patterns in inputs. No “right” or “wrong” answer.
- Reinforcement learning
  - Learner is not told ***which*** actions to take, but gets reward/punishment from environment and adjusts/learns the action to pick next time.

Biological Learning

- Which kind of learning is ecologically useful ?
- Supervised Learning
  - How often are animals given a neat pair of stimuli: Action $\rightarrow$ Result to learn? Perhaps in the lab but rarely otherwise
- Unsupervised learning
  - Useful for organising what is sensed in the world but not for choosing actions

RL in Nature

- Many actions that humans and animals make do not fall into neat action → reward pairs
- Sometimes reward (or punishment) comes some time after an action, or requires a chain of actions
- This is where RL is useful

What is RL?

- **Reinforcement Learning (RL)** is learning to act in order to maximize a future reward.
- RL is a class of tasks which require a trial-and-error learning
- We will talk about computer learning, using the term **agent** to refer to the computer / robot / program
- Features of RL:
  - Learning from rewards – sometimes rewards are rare or delayed
  - Interacting during the task (i.e. sequences of states, actions and rewards)
  - Exploitation/exploration trade-off
  - Problem of goal-directed learning

Example

- Playing poker when you don’t know the rules:
  - bet, bet, bet: you lose
  - bet, fold: you lose
  - bet, bet, bet: you win
  - ….
- Feed back is occasional, usually after a sequence of actions
- The task is to learn when to bet, and how much, to optimise winnings

Practical Applications

- Animal learning
  - e.g. animal learning to find food and avoid predators
- Robotics
  - e.g. robot trying to learn how to dock with charging station
- Games
  - e.g. chess player learning to beat opponent
- Control systems
  - e.g. Temperature thermostat keeping warmth, while minimising fuel consumption

State Space

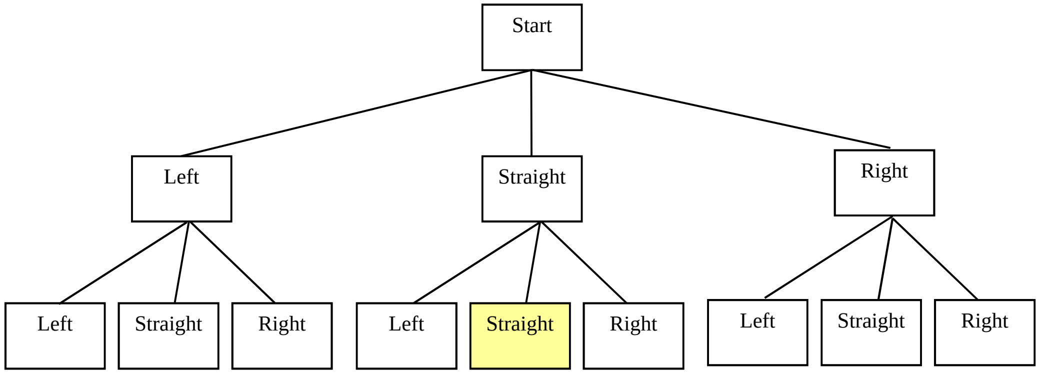
- Central to RL is the idea of **State Space**
- An agent occupies a given state at a given time
- To keep it simple, we will use a discrete state space – one where the agent moves from state to state in fixed steps, a bit like a chess board
- An action moves the agent from one state to another (states might be physical locations, but do not have to be).
- The state at time t is denoted s_t and the action taken at time t is denoted a_t

Representing States

- States might be physical locations, represented by coordinates
- They might be defined by the history of steps that took the agent to the current state
  - E.g. State Left,Straight,Right
- The latter can be less efficient as many series of steps can take you to the same location

State Trees

- You can think of all the possible states that result from a series of actions as a tree



The yellow leaf is the state you are in after going Straight, Straight.

With an equal number of choices at each step (*n*) and *d* steps,

there are n^d possible states you could be in.

Deterministic or Stochastic State Spaces

- An agent in state s_t might perform action a_t and move to state s_{t+1}
- A transition model tells the agent the new state given a current state and an action
- Without a transition model, the action must be taken for real and the new state is a physical state in the environment
- In a deterministic state space there is a function $T(s_t, a_t) = s_{t+1}$ that tells you the new state arrived at
- In a stochastic state space, there is a probability distribution that tells you the probability of each new state: $P(s' \mid a, s)$ tells you the probability of moving to state s' if you perform action a in state s

Stochastic Reward

- In the same way, reward may be deterministic or stochastic
- The reward for putting a pound in a slot machine is stochastic
- The reward for putting a pound in a chocolate vending machine is (usually!) deterministic

What to Learn?

- Utility based learning
  - Learn to relate **states** to **utility** – looks at all possible states that it might move to and picks the one with the highest utility
- Q-Learning
  - Learns the relationship between **actions** and **utility** so it can pick the action with the highest utility
- Reflex Learning
  - Learns to relate a **state** with an **action** – no utility is used

Utility Learning

- Requires a model of state transitions:
  - For each possible action:
    1. Predict the new state that it would take you to
    2. Look up the value of that state
    3. Choose the best
- Harder if the states are stochastic as you need to know the probability of each new state and its value
- “If I do this, I’ll be in that state, which will give me the best reward”.

Q Learning

- Learn the utility of each action in a given state directly
- No need for a model of the state transitions – just a model of the utilities of actions
- “If I do this, I don’t know what state I’ll be in, but my reward will be the best.”

Reflex Learning

- Learn actions from states
- No model of the state transitions needed
- No idea of utility needed
- Just look up your current state, see what the best action is, do it.
- “I don’t know what this action will lead to, or what reward it will bring, but I know it is the best thing to do.”

RL Policy

- However the agent learns, the rules that determine its actions as known as a policy
- **POLICY** $\pi_t(s, a) = P(a_t = a | s_t = s)$
- Given the state at time *t* is *s*, the policy gives the probability that the agent's action will be *a*.
- **Reinforcement learning => learn the policy**

Reward And Return

- The **reward function** indicates how good things are at this time instant
- But the agent wants to maximize reward in the long-run, i.e. over many time steps
- We refer to the reward of a whole policy as its **return**
- Calculating return:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T = \sum_{k=t+1}^{T} r_k$$

- where *T* is the last time step of the world.
- So, it is just the sum of all the rewards.

Discounted Return

- The geometrically discounted model of return:

$$R_t = r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \ldots + \gamma^T \cdot r_T = \sum_{k=0}^{T} \gamma^k \cdot r_{t+1+k}$$

- where $0 \leq \gamma \leq 1$ is the **discount rate**. (Gamma)
- Used to:
  - bound the infinite sum
  - Give more weight to earlier rewards (e.g. to give preference to shorter paths)

Value

- The Value of any state is its expected value over the entire policy
- During learning, an agent will try to bring its own measure of each state's value as close as possible to the true return that would be gained from that state:

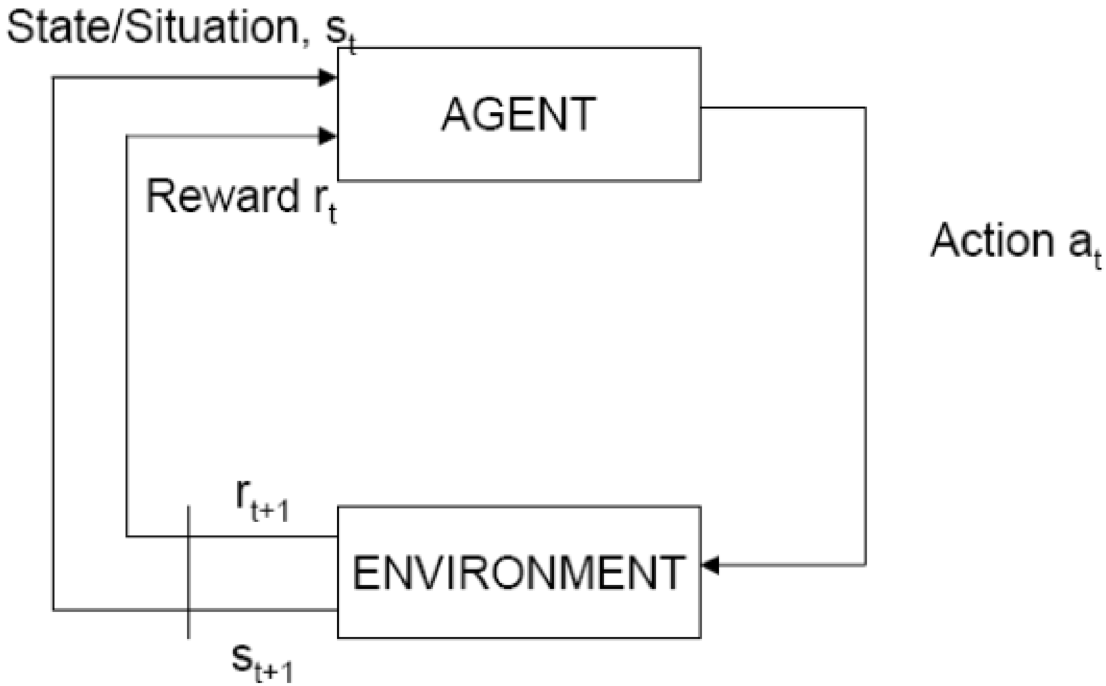
$$V(s_t) \rightarrow V(s_t) + \alpha[R_t - V(s_t)]$$

- Where α is the learning rate $0<\alpha<1$

Exploration / Exploitation

- Starting from zero, every action the agent takes is exploration
- After some time, it knows that one action is quite good.
  - Should it just stick to this action (exploitation)
  - Or look for a better one (exploration)
- The optimal strategy is hard to find, but starting with a lot of exploration and moving towards exploitation over time is sensible

RL Framework: How Does It Work?



1. Agent in **state** s_t chooses **action** a_t
2. World changes to state s_{t+1}
3. Agent perceives situation s_{t+1} and gets **reward** r_{t+1}

The Learning Process

- In the current state:
  - Collect the reward for this state
  - Update the value for the state using the learning rule $V(s_t) \rightarrow V(s_t) + \alpha[R_t - V(s_t)]$
  - Keep going until $V(s_t) = R_t$
- A problem occurs when rewards only come at the end of a set of actions, with steps preceding the goal having zero reward

Temporal Difference Learning

- The solution is to assume that the value of any state is similar to the value of its neighbouring states as they can be easily visited from the current state
- So, to update the current state’s value, look at the reward you get and the value of the next state and update the current state thus:

$$V(s_t) \rightarrow V(s_t) + \alpha[R_t + \gamma V(s_{t+1}) - V(s_t)]$$

- α is the learning rate, γ is the discount term

Long Searches

- What if the next state's reward is zero too?
- And the next, and the next?
- Well, in the end, a reward (perhaps the goal state) will be found
  - The state that led to it will now have value, giving two states to look for
  - Repeat enough and you will have a value for every possible state
  - Or the world will end, which ever happens first

Representing the Values

- Simple RL learning represents the values in a huge table
- In any task of real use, this table would be far too big and would never be completely filled
- Many states would never be visited, and when they finally were, there would be no action in the policy table

Neural Networks and RL

- Instead of using a giant table, the values and states can be learned using a neural network
- Now the agent learns a function between state and value (or action) that has two advantages:
  - It is smaller and faster than a table
  - It can generalise to states it hasn't seen before