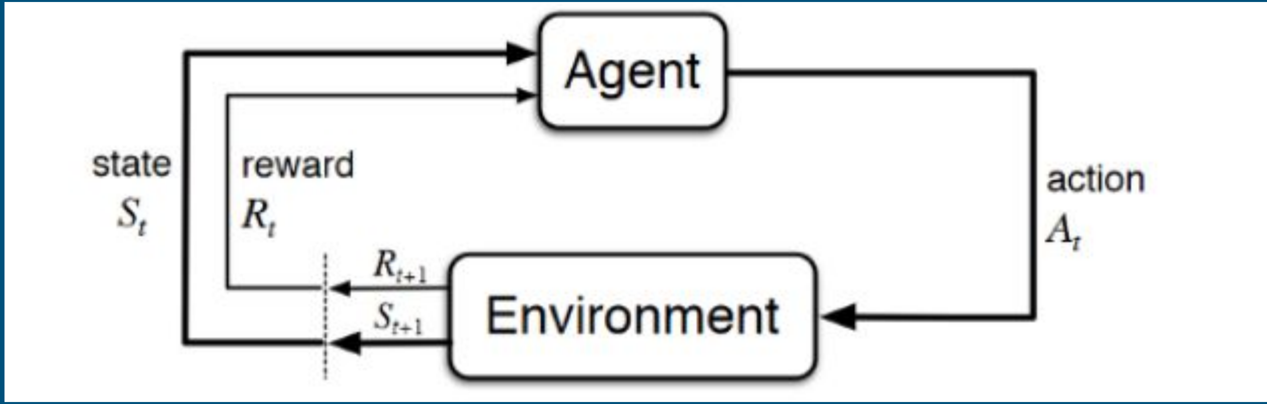
Reinforcement Learning

the environment is your guide

Machine Learning

- Supervised learning: y = f(x) when you have both x and y
- Self-supervised learning: y = f(x) when y is somehow derived from x
- Unsupervised learning: y = f(x) when you only have x
- Reinforcement learning: y = f(x) you can have both x and y, or you can have neither, but either way you have a *decision* to make.
    - Used in gaming, robotics, healthcare, finance, dialogue systems, traffic control, recommendation engines, self-driving cars, adaptive systems, ….

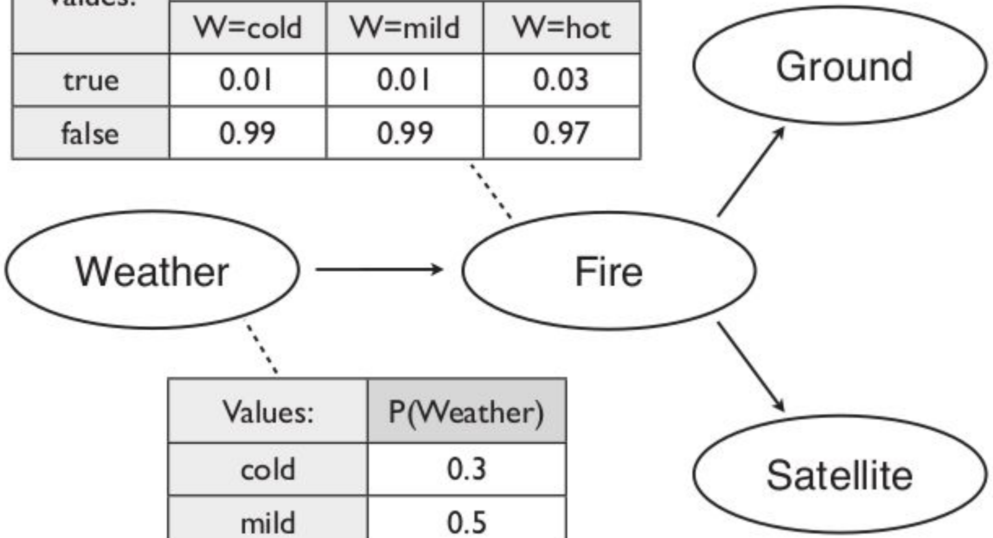
RL: Intuition



http://incompleteideas.net/book/bookdraft2017nov5.pdf

Example (Lison 2014, Chapter 3)

Values:	P(Fire|Weather)		
	W=cold	W=mild	W=hot
true	0.01	0.01	0.03
false	0.99	0.99	0.97

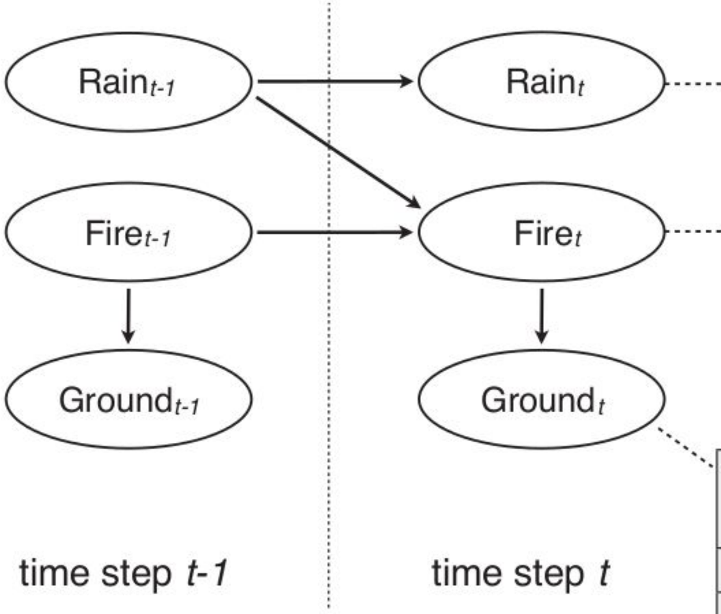


Values:	P(Weather)
cold	0.3
mild	0.5
hot	0.2

Values	P(Ground|Fire)	
	F=true	F=false
alarm	0.9	0.05
quiet	0.1	0.95

Values	P(Satellite|Fire)	
	F=true	F=false
alarm	0.7	0.01
quiet	0.3	0.99

Example, now time series



Values	$P(Rain_t \mid Rain_{t-1})$	
	R_{t-1}=true	R_{t-1}=false
true	0.7	0.3
false	0.3	0.7

Values	$P(Fire_t \mid Rain_{t-1}, Fire_{t-1})$			
	R_{t-1}=true		R_{t-1}=false	
	F_{t-1}=true	F_{t-1}=false	F_{t-1}=true	F_{t-1}=false
true	0.4	0	0.9	0.01
false	0.6	1	0.1	0.99

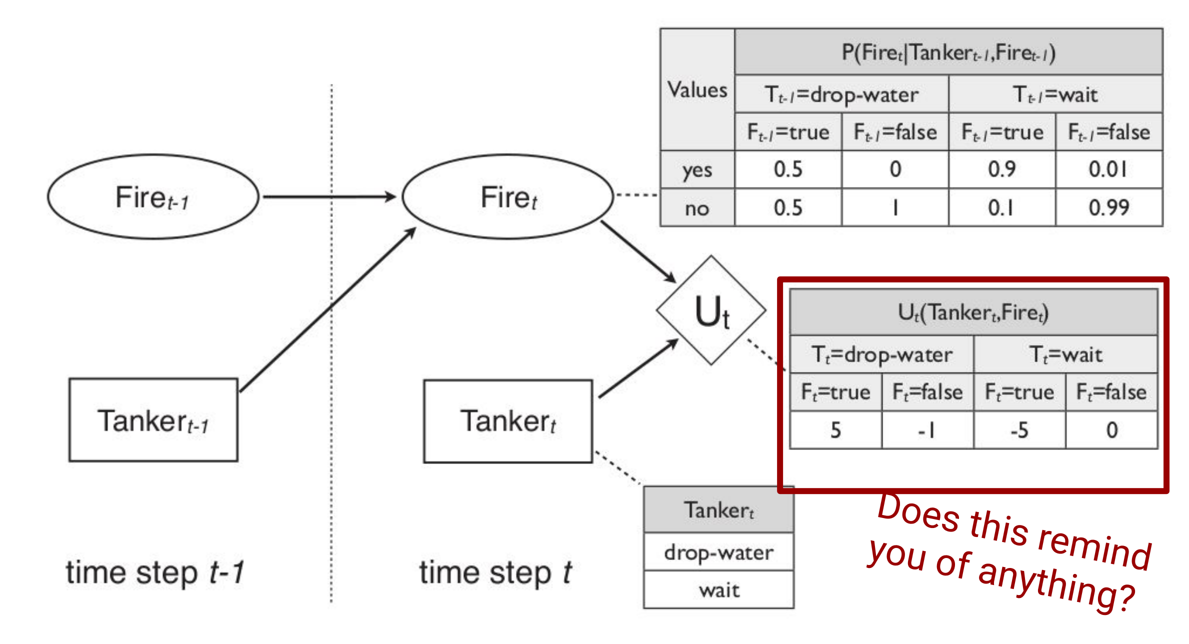
Values	$P(Ground_t \mid Fire_t)$	
	F_t=true	F_t=false
alarm	0.9	0.05
quiet	0.1	0.95

Example, now with a decision to make

Chance nodes are associated with conditional probability distributions that define the relative probabilities of the node values given the values in the parent nodes.

Decision nodes correspond to variables that are under the control of the system. The values of these nodes reflect an active choice made by the system to execute particular actions.

Utility nodes express the utilities (from the system's point of view) associated with particular situations expressed in the node parents. Typically, these parents combine both chance and decision variables.



Learning: estimating the probabilities

- Treat each factor independently and estimate each one individually
- Can use Maximum likelihood estimation (MLE)
  - What are some possible issues with this?
  - Example: You have only observed once that Weather=cold, then what will you model predict?
- Can use Bayesian learning
  - What is needed to make this happen?
  - Example: You don't have any data, but you set a prior P(x) to something "sensible" like cold=0.4, hot=0.4, and warm=0.2
  - Then use new observations to update priors, likelihoods, and posteriors

Learning: estimating the probabilities

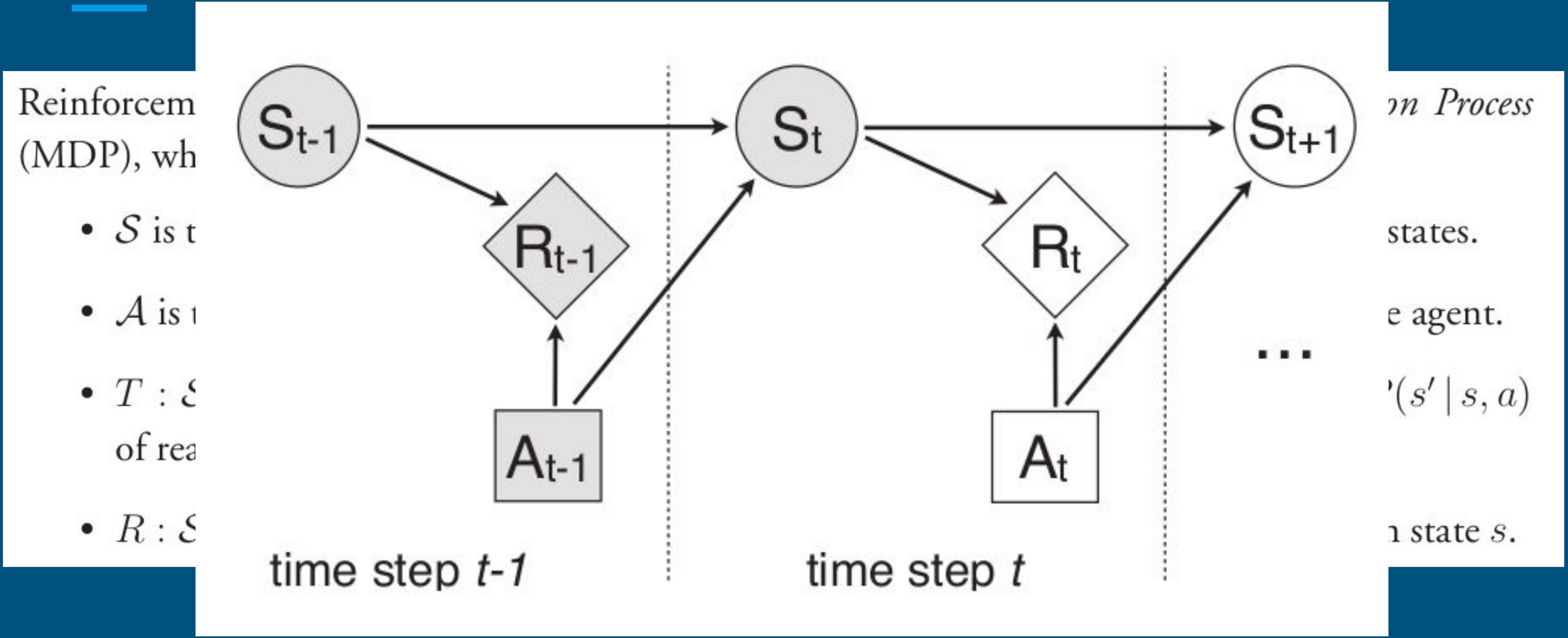
- Discrete/categorical data: just count things
- Continuous data: which distributions do I use for likelihoods P(x|y), priors P(y), and posteriors P(y|x)?
  - Rule of thumb: try to use distributions from the same family (conjugate priors)
  - Normal distribution: just need means and standard deviations, but estimation using Bayesian learning usually means you start with a value for each then update as data becomes available
  - Dirichlet (to help with categorical data): a continuous, multivariate distribution, need to estimate alpha parameters
- Reaching all of the states: explore, exploit

Markov Decision Processes (MDP)

Reinforcement learning tasks are typically based on the definition of a *Markov Decision Process* (MDP), which is a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ where:

- $\mathcal{S}$ is the state space of the domain and represents the set of all (mutually exclusive) states.
- $\mathcal{A}$ is the action space and represents the possible actions that can be executed by the agent.
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function and encodes the probability $P(s' \mid s, a)$ of reaching state s' after executing action a in state s.
- $R : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the reward function associated with the execution of action a in state s.

Markov Decision Processes (MDP)



Temporal-difference: Q-Learning

Goal: find the best function Q(s,a) that estimates the MDP (e.g., think of chess)

state

action

learning rate

$$Q_{k+1}(s_t, a_t) \leftarrow Q_k(s_t, a_t) + \alpha \left[r_{t+1} + \gamma\, Q_k(s_{t+1}, a_{t+1}) - Q_k(s_t, a_t)\right]$$

estimated Qk for new state

update Q for k+1 from what was known before in k

reward

discount factor: worth of future rewards vs. present rewards

Partially Observable MDP (POMDP)

$$\langle \mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, Z \rangle.$$

O: the set of possible observations

Z defines the probability $P(o \mid s)$ of observing o in the current state s.

How to make a RL agent?

- Need a way to represent states (initial state values, too)
- Need a way to model the factored joint distribution
    - Each factor needs its own prior, likelihood, posterior distribution
- Need a way to map from states to actions
    - And rewards!
- Easy to author different things (e.g., chess, dialogue)
- Works in live settings as well as from data or simulation
- Answer: opendial!