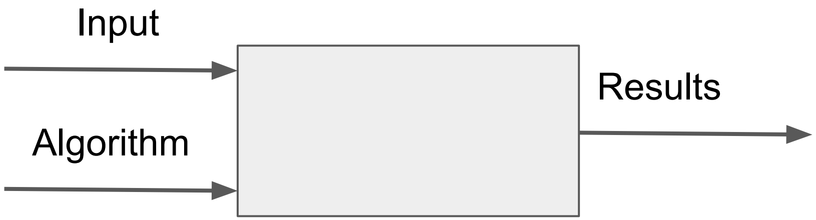
TX00DQ05-3001 Reinforcement Learning

peter.hjort@metropolia.fi

Machine learning

A branch (an active and quite large one) of artificial intelligence that deals with constructing systems where the focus is on **learning** from **available data** or **reactions of the environment**. Much of machine learning is built on **statistics** and (approximate) **optimization**.

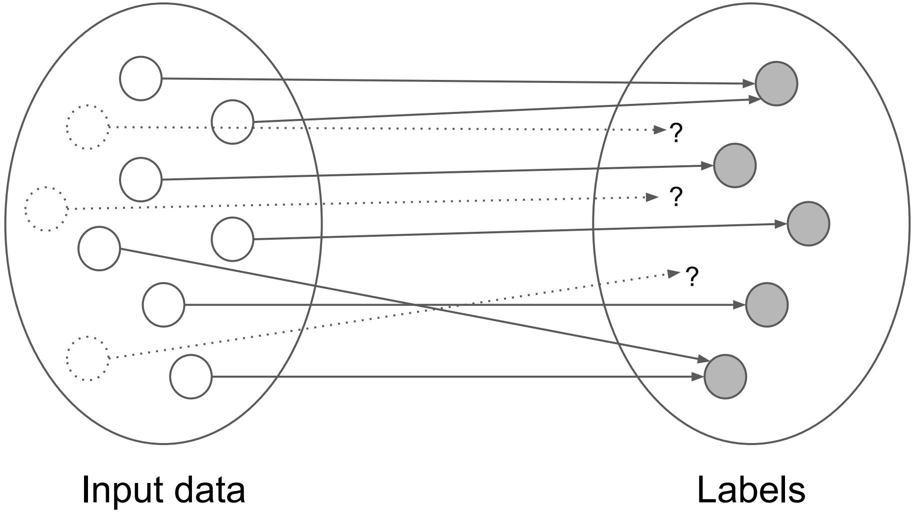


Traditional programming. Human invents how results are computed from data.

(Supervised) machine learning. System learns how input and results are related. The “algorithm” can be used as a component in an “intelligent” system.

Supervised learning

Starting point in supervised learning is that there is a set of data, and for each item in the data set a value (label) associated with that item. The target is to learn the function from input data to labels so that labels for unlabeled data items can be predicted.

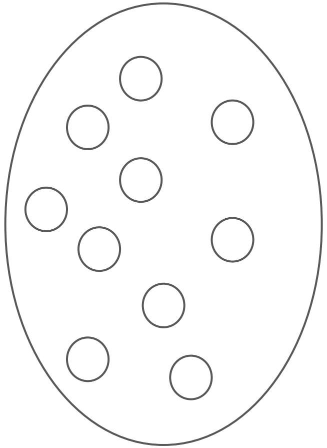


Examples of supervised learning applications:

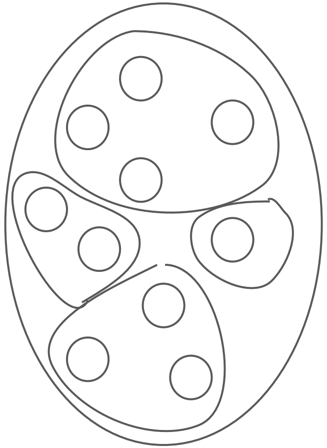
- Predicting house prices based on room count, room area, zip code etc.
- Labeling images into categories (is it a dog or is it a cat?)
- Grouping customers into categories based on buying behaviour

Unsupervised learning

Starting point is a set of unlabeled data. Target is to find relevant associations in the data without any external help (apart from selecting the algorithms to be used etc).



Input data



Clustered view

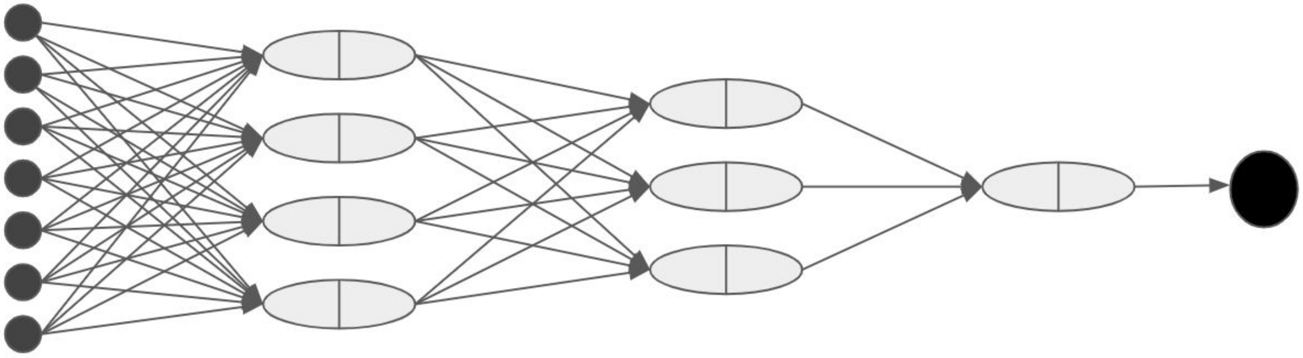
Examples of unsupervised learning:

- Clustering; group together data items that share some similarity
- Explain data by find out statistical distributions from which they can be combined from (mixed models)
- Find some alternative (latent) representation for the data (autoencoders, compression)

Deep learning

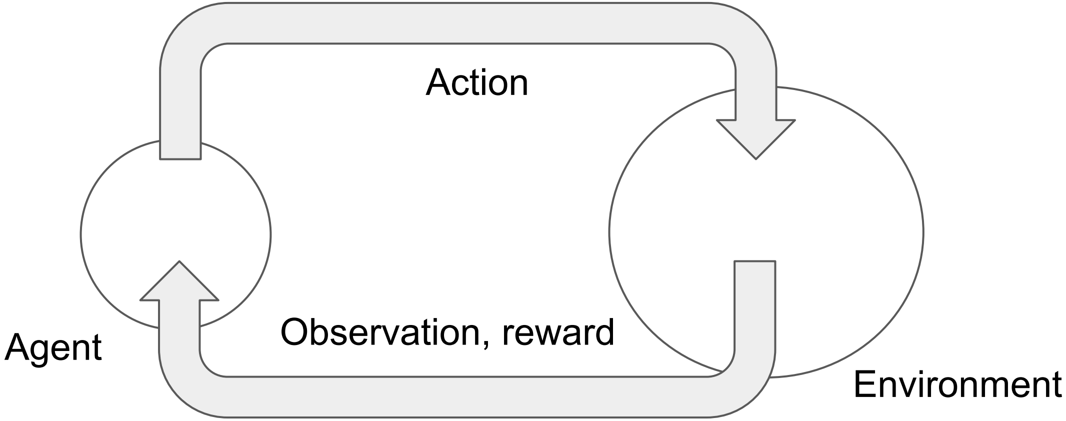
Supervised or unsupervised learning where the model has far more parameters than in shallow models such as linear regression, random forests, svms etc.

Deep models have typically hierarchical structure, reflecting their capability to identify hierarchical features and representations. Deep models in general are able learn wider families of functions to map from input to output.



Reinforcement learning

In reinforcement learning (RL) the set-up is more general and dynamic than in supervised / unsupervised learning variants. Key concepts in RL are an **agent** (the system being developed), **environment** (abstraction of the surroundings of the agent), and **rewards** that guide the agent to learn to perform the right **actions ie. make the right decisions**. Note that **the agent does not initially know the environment**, this is key aspect of reinforcement learning.



Some RL application areas:

- Robotics
- Online ad selection
- Learning to play games (Atari from pixel level, Chess, Go)

Reinforcement learning in context

In computer science associated (like we do) with **Machine Learning**.

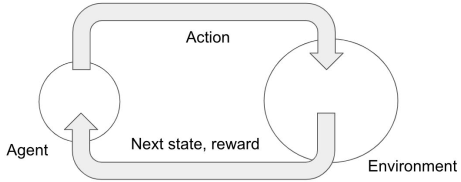
In mathematics the closest field is **Operations Research**.

In engineering **Optimal Control**.

In neuroscience closely related to **Reward (Dopamine) System**.

In psychology **Conditioning**.

Key concepts in reinforcement learning



- **Reward signal**, a real number, that guides the agent towards desired behaviour. There is not other guidance, so the agent has to learn by interacting with the environment without knowing how the environment operates, using **trial-and-error**. The reward signal might be, and usually is, **delayed.**
- Agent can influence the environment by taking **actions**.
- Process is **sequential**, ie. proceeds step by step. In one timestep the agent performs an action, receives reward and moves to next state. Often the term **sequential decision making** is used.
- The observed data (states) are not **i.i.d** (independent and identically distributed) but have strong correlations (state depends on previous state). This is in stark contrast with supervised learning.

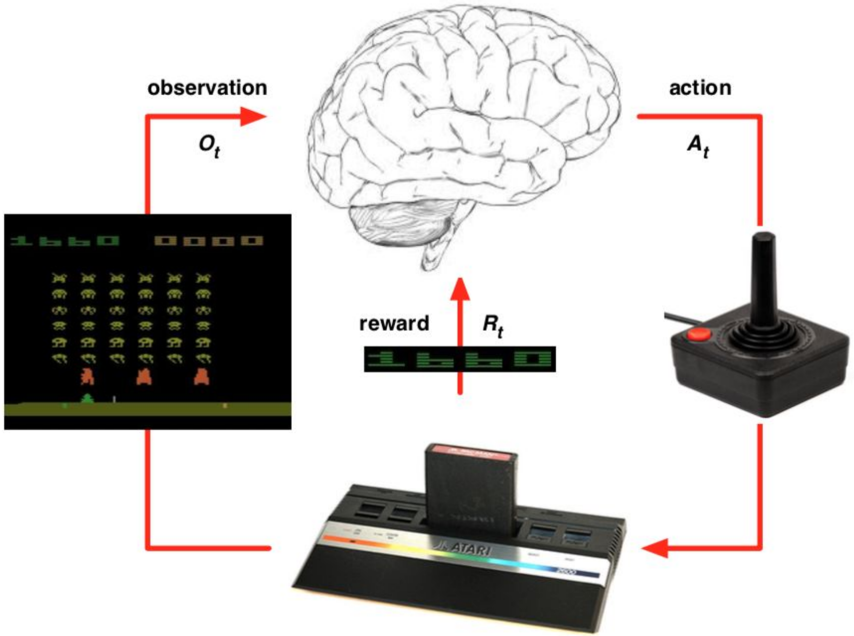
Examples of reinforcement learning

- A master chess player makes a move.
- An adaptive controller adjusts parameters of a petroleum refinery operation in real time.
- A gazelle calf struggles to its feet minutes after being born.
- A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station.
- An advert management system deciding which ad to place to a page next.

Quadrotor https://www.youtube.com/watch?v=T0A9voXzhng

Autonomous helicopter https://www.youtube.com/watch?v=VCdxqn0fcnE

Atari Example: Reinforcement Learning



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

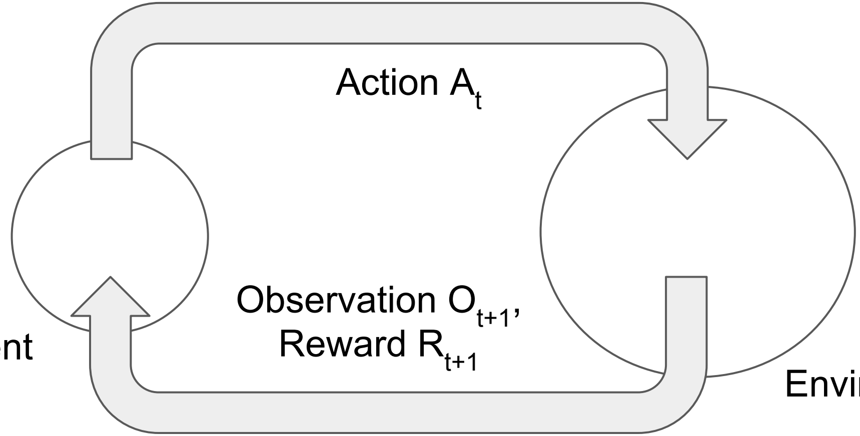
From RL slides by David Silver.

Atari examples

Atari video https://www.youtube.com/watch?v=V1eYniJ0Rnk

Pacman https://www.youtube.com/watch?v=4MlZncshy1Q

Observation, reward, action cycle



Agent

Environment

At time step *t*:

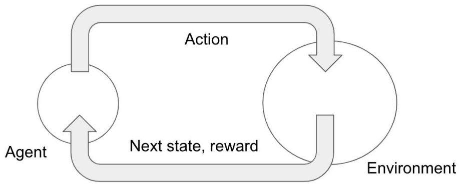
Agent

- Performs action A_t
- Receives observation O_t
- Receives reward R_t

Environment

- Receives action A_t
- Sends observation O_{t+1}
- Sends reward R_{t+1}

Goal in reinforcement learning



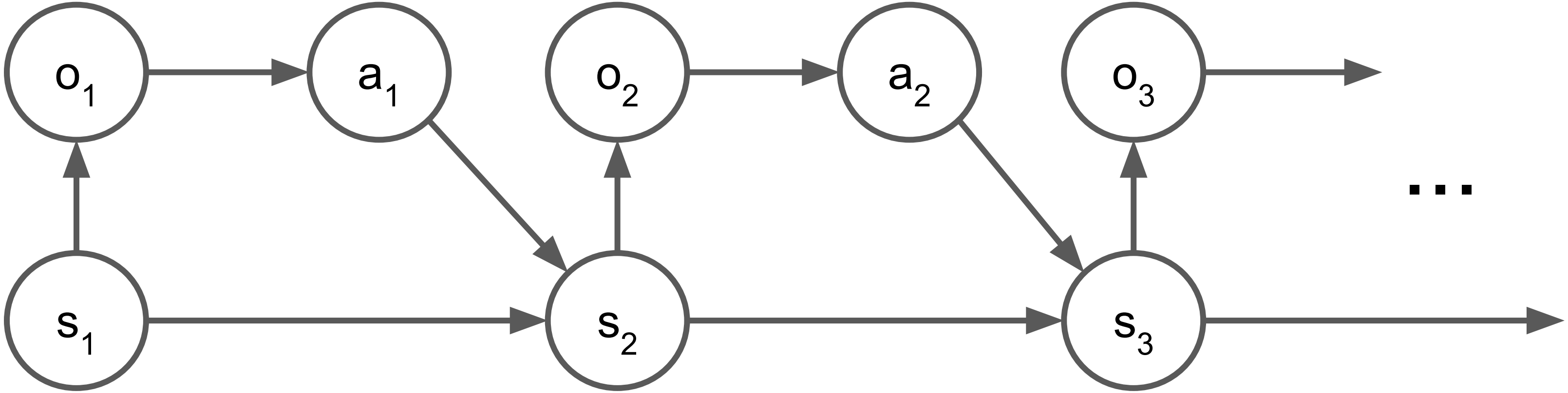
The agent receives from the environment a reward signal (a real number) R_t at every timestep *t*.

The assumption in reinforcement learning is that any **goal for the agent** can be expressed as **maximisation of the cumulative feedback**, ie. find the actions to perform to maximise the accumulated reward signal. Note that the accumulation is not necessarily sum of all rewards, often discounting factor is used.

Cumulation of rewards drives the agent not consider only immediate rewards (act greedily), but optimise long-term reward.

Environment state and observations

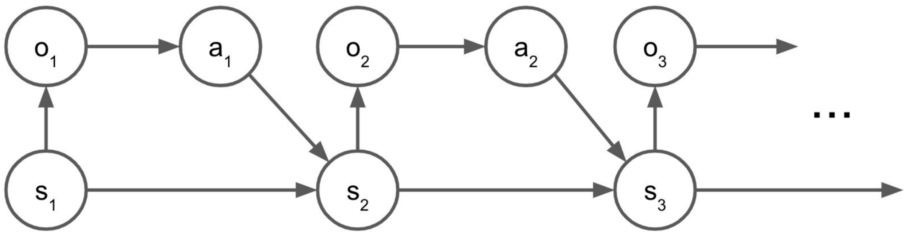
The observations O_t available to the agent capture only part of the environment state S_t. Agent actions A_t are based on observations, and together with the previous state of the environment s_t, change the environment state to s_{t+1}.



Fully vs. partially observable environment

Fully observable environment: The agent knows everything about the environment, ie. the state of the environment is directly accessible. In this case $O_t = S_t$.

Partially observable environment: The agent only knows the observations, environment has hidden information that will have an effect on transition from S_t to S_{t+1}.



Modeling the environment - Markov state

We say that the state model is **Markovian** iff (if and only if) the probability function for entering the next state S_{t+1} from state S_t satisfies

$$p(S_{t+1} \mid S_t) = p(S_{t+1} \mid S_1, S_2, \dots , S_t)$$

In other words, **probability of entering the next state depends only on current state**, not on states before the current state.

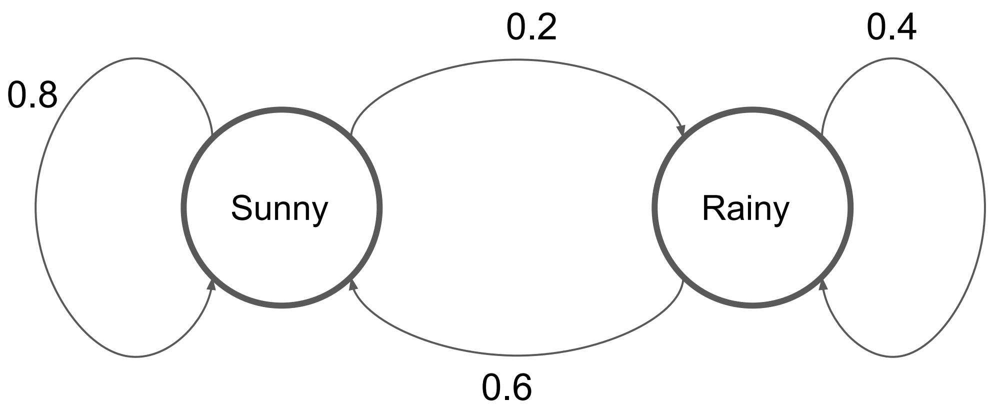
So, the current state captures everything there is to know - we need not care about the history.

The environment state in reinforcement learning is often/usually assumed to be Markovian.

Markov process

Markov process is a tuple <S, P>, where

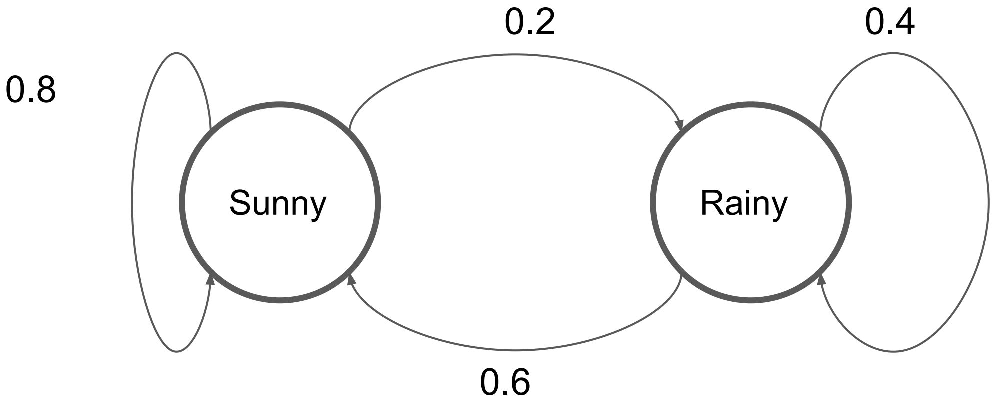
- S is the set of **states**
- P is the **(transition) probability function** $p(s, s') = p(S_{t+1} = s' \mid S_t = s)$



S = { Sunny, Rainy }

p(Sunny, Sunny) = 0.8
p(Sunny, Rainy) = 0.2
p(Rainy, Rainy) = 0.4
p(Rainy, Sunny) = 0.6

Transition matrix representation



S = { Sunny, Rainy }

p(Sunny, Sunny) = 0.8
p(Sunny, Rainy) = 0.2
p(Rainy, Rainy) = 0.4
p(Rainy, Sunny) = 0.6

Transition matrix P consists of the transition probabilities where rows denote the current state and columns denote the next state. Row sums are = 1.

	Sunny	Rainy
Sunny	0.8	0.2
Rainy	0.6	0.4

Running the Markov process

If the weather today is Rainy, how is the weather going to tomorrow? How about day after? How about in average?

```
import numpy as np
import numpy.linalg as LA

P = np.array(([0.2, 0.8], [0.6, 0.4]))
t0 = np.array([0.0, 1.0])

t1 = np.dot(t0, P)
print(t1)
t2 = np.dot(t1, P)
print(t2)
print(np.dot(t0, LA.matrix_power(P, 2)))
print(np.dot(t0, LA.matrix_power(P, 10)))
print(np.dot(t0, LA.matrix_power(P, 30)))
print(np.dot(t0, LA.matrix_power(P, 50)))
```

```
[0.6 0.4]

[0.36 0.64]
[0.36 0.64]
[0.42852649 0.57147351]
[0.42857143 0.57142857]
[0.42857143 0.57142857]
```

Markov reward process

Markov reward process is a tuple <S, P, R, γ>, where

- S is the set of **states**
- P is the **probability function** $p(s, s') = p(S_{t+1} = s' \mid S_t = s)$
- R is the **reward function** $R(s) = \mathbb{E}(R_{t+1} \mid S_t = s)$
- γ is the **discount factor** in range 0..1

Note: reward function is defined as an expectation, ie. we consider all possible rewards in a state weighted by their probabilities.

Reward examples

- One positive reward when the target has been achieved
- Intermediate rewards and a reward in the end - need to potentially balance between intermediate and end but gives the agent some direction
- Negative reward per step - give incentive to reaching the target fast

Return and discount factor

Return G_t at timestep t is the sum of all rewards after step t:

$$G_t \stackrel{\text{def}}{=} R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \ldots$$

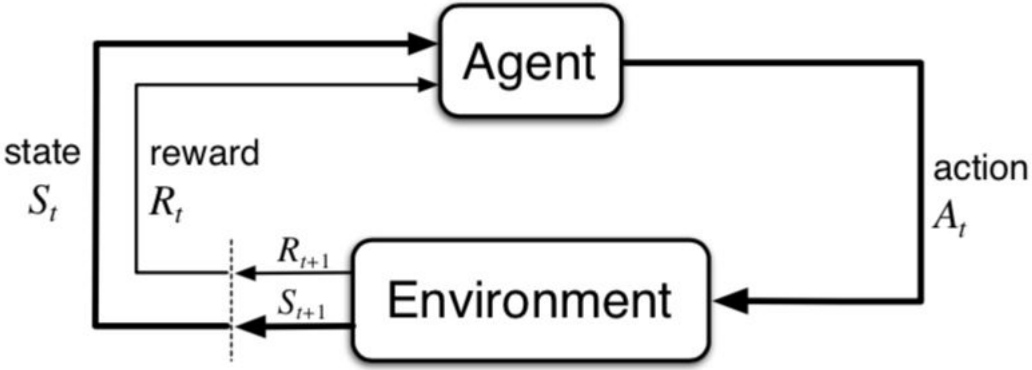
Where γ is in range 0..1 (inclusive) is the discount factor.

Discount factor ($\gamma < 1$) is used for reducing the weight of future rewards in the return. This reflects the idea that future rewards are risky (compare with net present value calculation in finance).

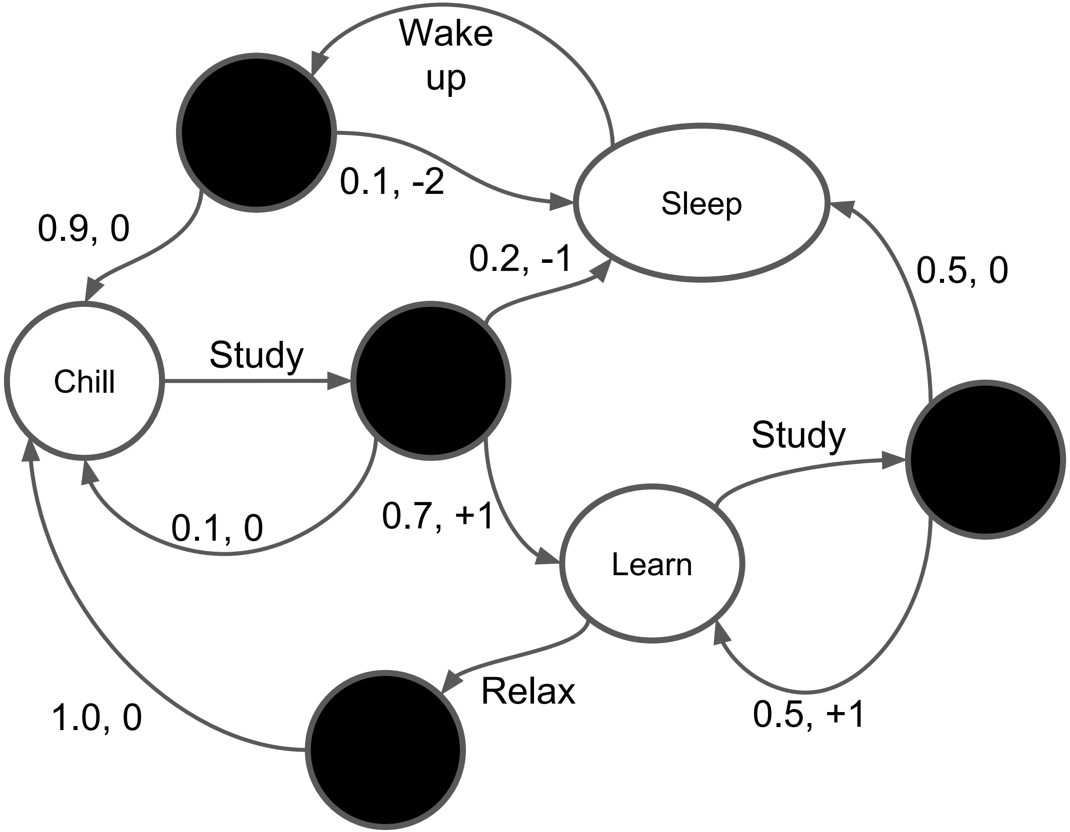
Markov decision process

A tuple <*S, A, P*, γ> is a Markov decision process where

- *S* is the set of states, $S = \{ S_t , t = 0, 1, 2, ... \}$
- *A* is the set of actions $A = \{ A_t, t = 0, 1, 2, … \}$
- *P* is the transition function $p(s', r \mid s, a) \stackrel{def}{=} Pr\{ s' = S_t , r = R_t \mid s = S_{t-1} , a = A_t \}$
- γ is the discount factor



MDP example: Student life



S = { Chill, Learn, Sleep }
A = { Study, Wake up, Relax }

p(Sleep, -1 | Chill, Study) = 0.2
p(Learn, +1 | Chill, Study) = 0.7
p(Chill, 0 | Chill, Study) = 0.1

p(Study, +1 | Learn, Study) = 0.5
p(Sleep, 0 | Learn, Study) = 0.5

p(Chill, 0 | Learn, Relax) = 1.0

…

p(Learn, * | Sleep *) = 0.0

Some properties of the transition function

For given state *s* and action *a* the probabilities of next states *s'* and rewards *r* sum up to 1:

$\sum_{s' \text{ in } S} \sum_{r \text{ in } R} p(s', r \mid s, a) = 1$

State-transition probabilities can be computed like this:

$p(s' \mid s, a) = \sum_{r \text{ in } R} p(s', r \mid s, a)$

Expected rewards for state-action pairs:

$r(s, a) = \sum_{r \text{ in } R} r \sum_{s' \text{ in } S} p(s', r \mid s, a)$

Supporting material

David Silver RL lecture 1: https://www.youtube.com/watch?v=2pWv7GOvuf0 from 6:25 onwards

Pieter Abbeel on RL for robotics: https://www.youtube.com/watch?v=evq4p1zhS7Q

https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690

(Sergey Levine UC Berkeley Deep Reinforcement Learning course lecture 1: https://www.youtube.com/watch?v=opaBjK4TfLc&list=PLkFD6_40KJIxJMR-j5A1mkxK26gh_qg37&index=25 from 13:52 onwards.)

Exercises (session01b on Thu 21st Mar)

session01b.ipynb

Install gym; https://github.com/openai/gym (and keras-rl; https://github.com/keras-rl/keras-rl) (or something similar) and try out the simple cartpole example. Take a look at one of the gym agents, for example random_agent.py. Does the agent have any high-level similarities with the agent/environment structure we have discussed? A bit of writing on gym: https://arxiv.org/pdf/1606.01540.pdf

Offline: watch talk given by Pieter Abbeel (https://www.youtube.com/watch?v=evq4p1zhS7Q, about 20mins). Was the video in general understandable? What were the hard parts? What were the most interesting concepts for you? Any other thoughts on the topics in the video?

Exam 27.3

session01.pdf

Sutton & Barto: 1.1 - 1.6, 3.1 - 3.3