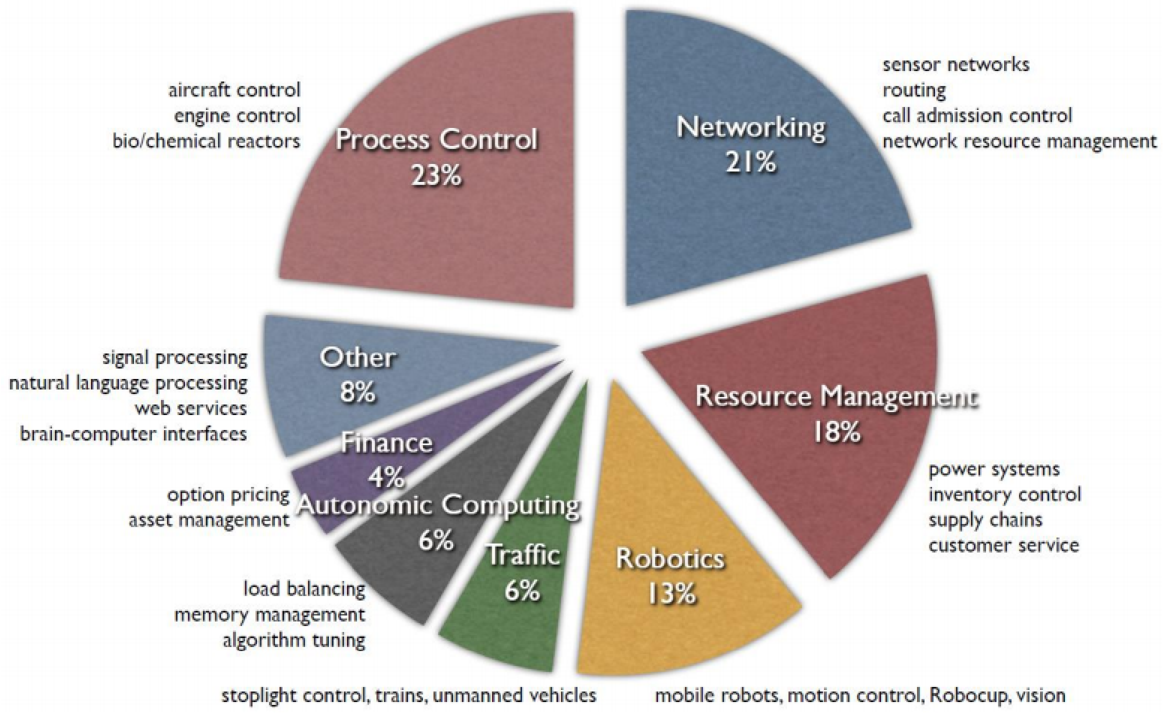
Reinforcement Learning

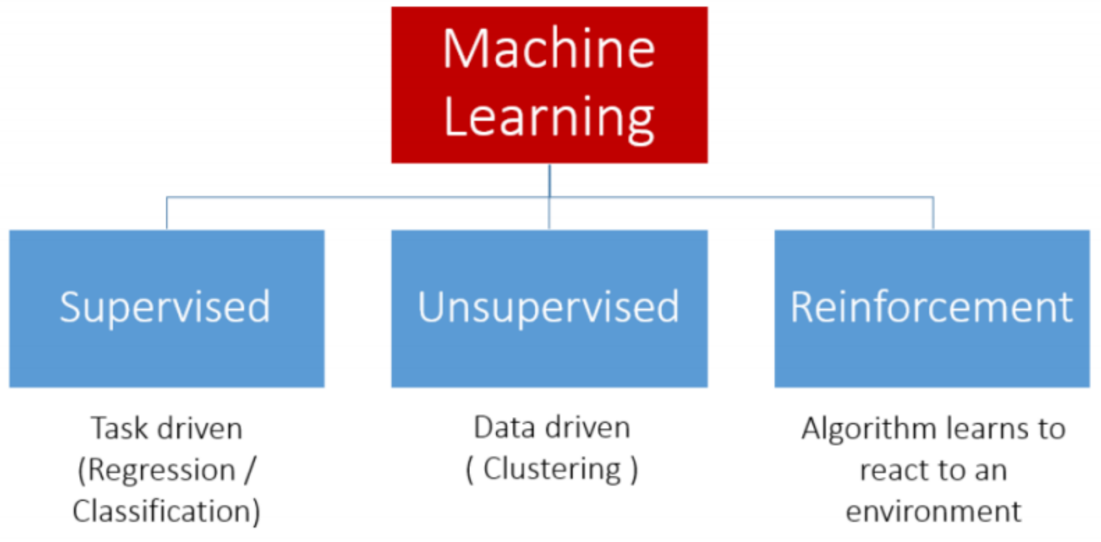
• Terminology

1. What is an agent?
   An entity interacting within an environment according to a policy
2. What is an environment?
   The entity with which the agent interact, afrom which the agent receives feedbacks and reward
3. What is a state?
   A descriptor of the environment and the agent (a sufficient statistics of the history for the agent's decision)
4. What is an action?
   A way for the agent to interact with the environment
5. What is a policy?
   A rule to choose actions given states
6. What is a reward?
   A signal denoting how good the current action, and implicitly, the preceding ones were

- Applications of RL:



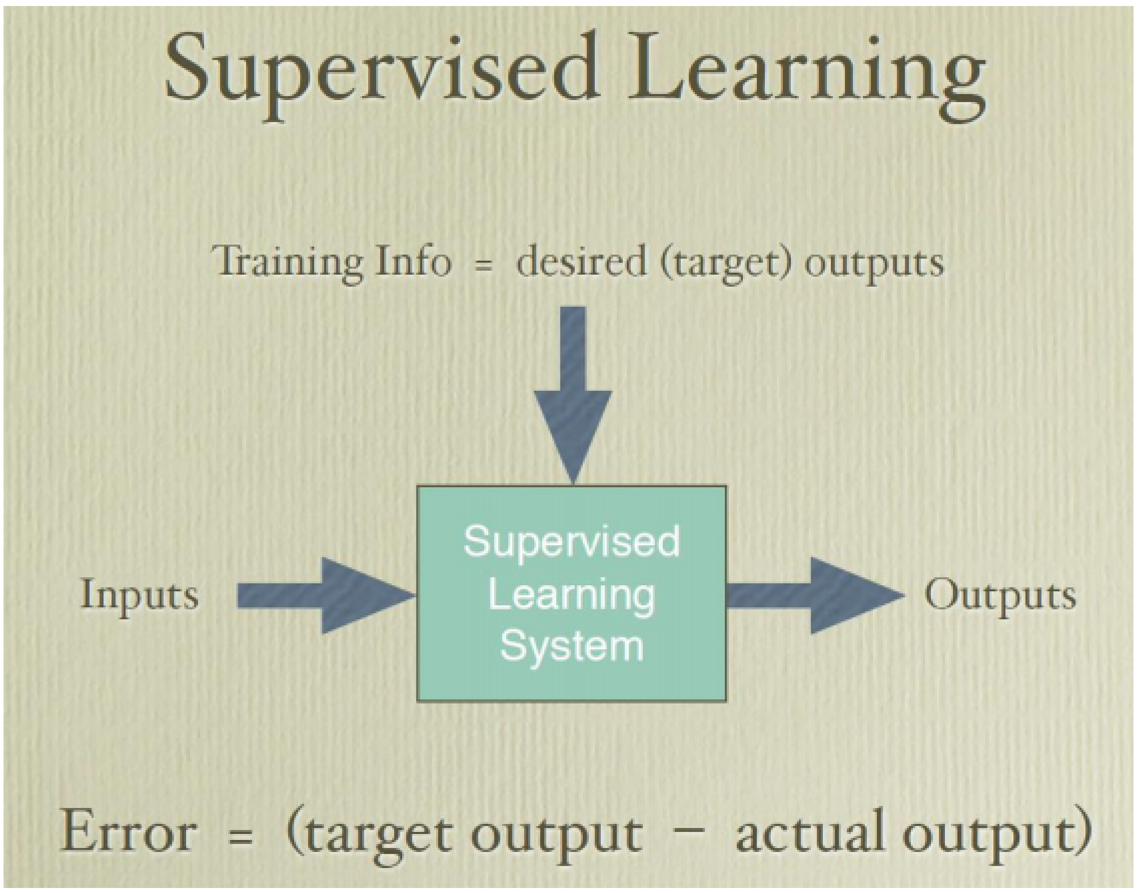
• Types of machine learning


Machine Learning
Supervised
Unsupervised
Reinforcement
Task driven (Regression / Classification)
Data driven (Clustering)
Algorithm learns to react to an environment

Supervised Learning

We send in inputs through the machine learning model (Supervised Learning System) and we get output.

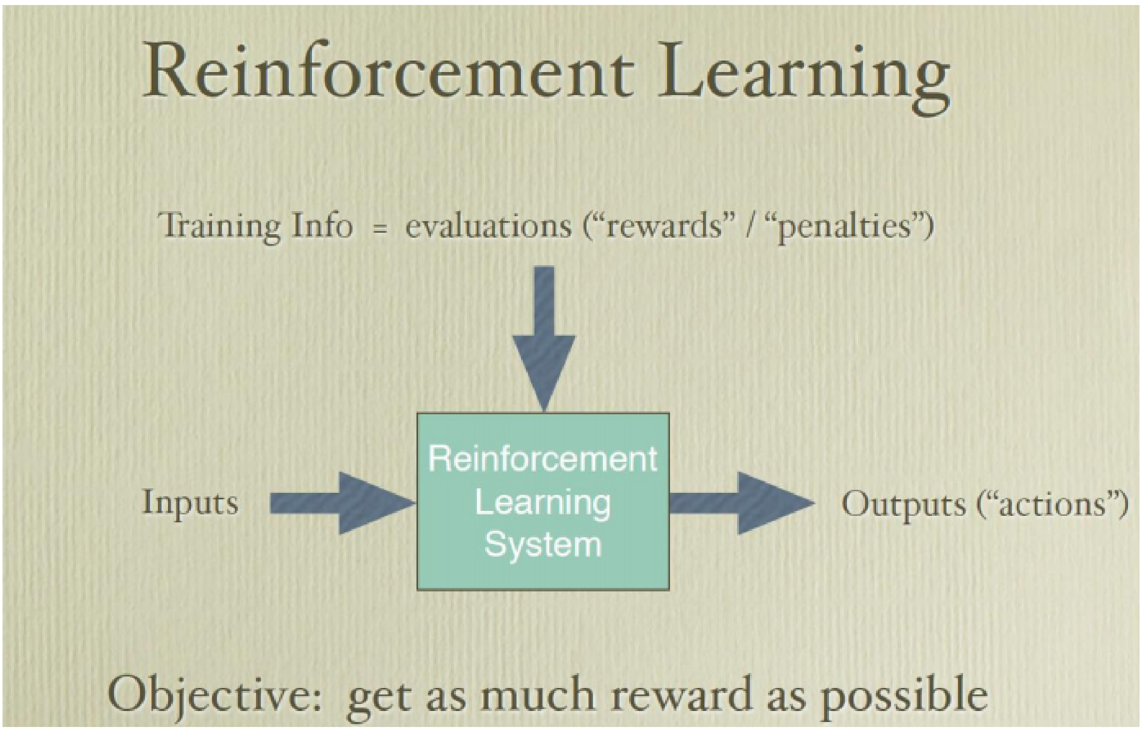
The training fixes and adjusts the system and its weights according to the desired (target) outputs, which is the ***training info*** .


Supervised Learning
Training Info = desired (target) outputs
Inputs
Supervised Learning System
Outputs
Error = (target output − actual output)

Reinforcement Learning

The machine learning model is based on evaluation by “rewards” and “penalties”.

Objective: Get as much reward as possible


Reinforcement Learning
Training Info = evaluations ("rewards" / "penalties")
Inputs
Reinforcement Learning System
Outputs ("actions")
Objective: get as much reward as possible

• The reinforcement learning problem

State, Action and Reward

- Learning is guided by the reward

- An infrequent numerical feedback indicating how well we are doing
- Problems:
    - The reward doesn't tell us what we should have done!
    - The reward may be delayed - doesn't always indicate when we made a mistake

The *reward* function

- Corresponds to the fitness function of an evolutionary algorithm
- r_{t+1} is a function of (s_t, a_t)
- The reward is a numeric value. Can be negative ("punishment").
- Can be given throughout the learning episode, or only in the end.
- **Goal**: Maximize the total reward

Maximizing total reward

Future rewards may be uncertain. We might care more about the rewards that come sooner. Therefore, we discount future rewards.

A good strategy for an agent would be to always choose an action that **maximizes the (discounted) future reward.**

Action Selection

At each learning stage, the RL algorithm looks at the **possible actions** and calculates the **expected average reward**.

An action will be selected using:

- Greedy strategy: pure exploitation
- Greedy strategy: exploitation with a little exploration
- Soft-Max strategy

Policy (π) and Value (V)

- The set of actions we took define our policy (π)
- The expected rewards we get in return, defines our value (V)

Markov Decision Process

If we only need to know the **current state**, the problem has the ***Markov property*** .

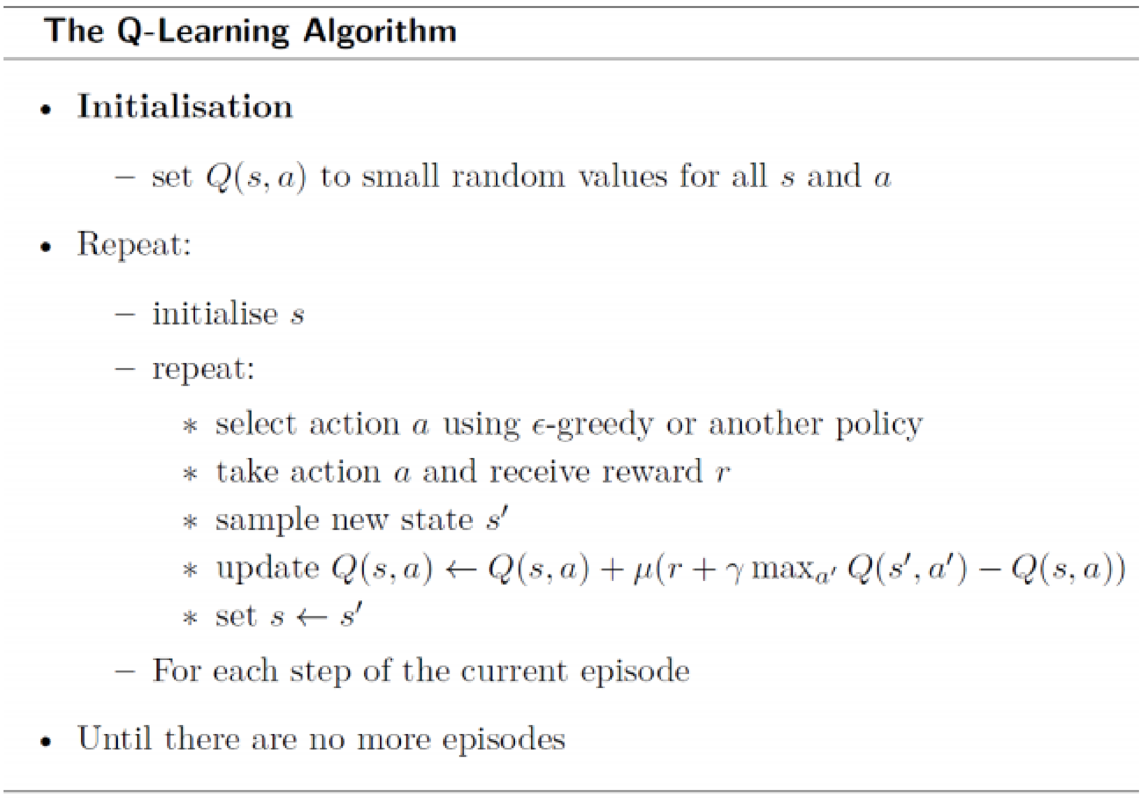
Value

The expected future reward is known as the ***value.***

Two ways to compute the value:

- The value of a state, $V(s)$, averaged over all possible actions in that state ***(state-value function)*** .
- The value of a state/action pair ***(action-value function)***
- Q and V are initially unknown, and learned iteratively as we gain experience.

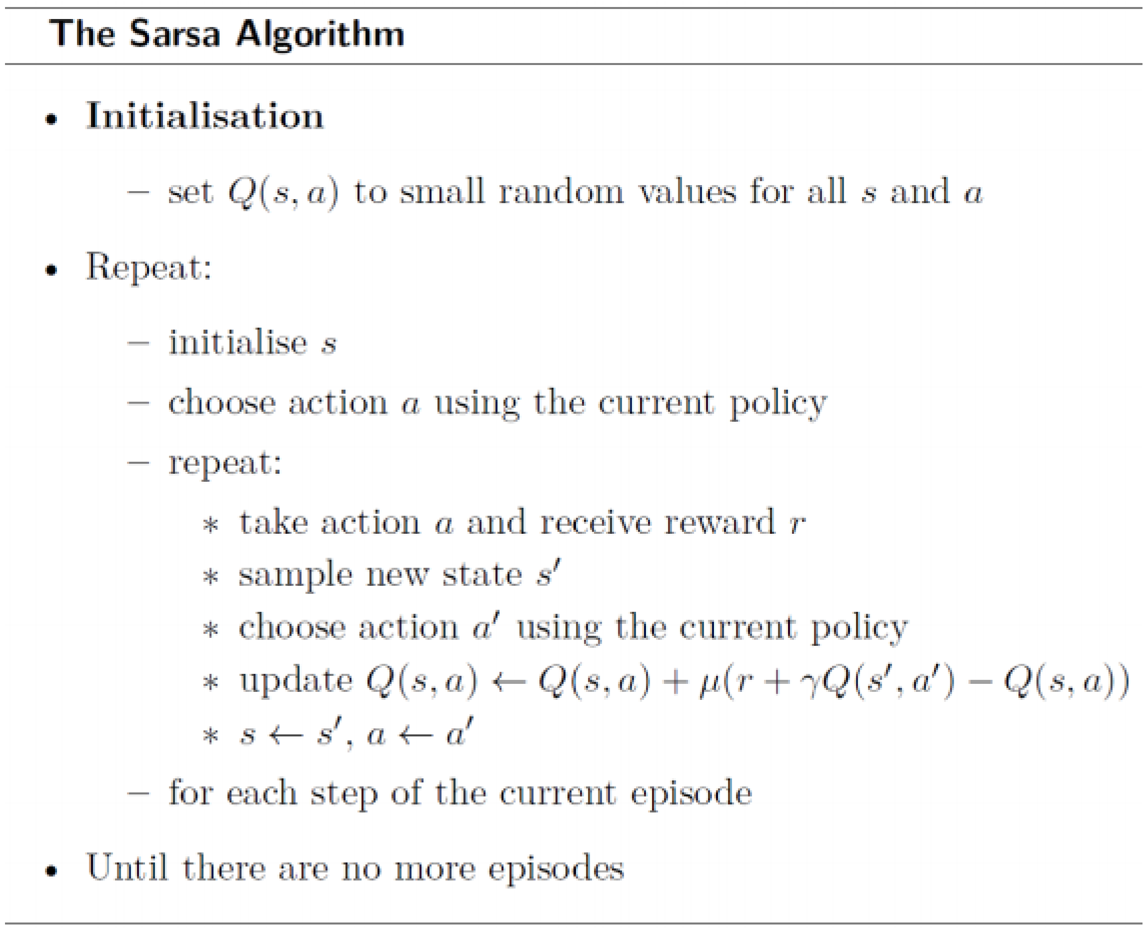
The Q-Learning Algorithm



The Q-Learning Algorithm

```
• Initialisation
    – set $Q(s, a)$ to small random values for all $s$ and $a$
• Repeat:
    – initialise $s$
    – repeat:
        ∗ select action $a$ using $\epsilon$-greedy or another policy
        ∗ take action $a$ and receive reward $r$
        ∗ sample new state $s'$
        ∗ update $Q(s, a) \leftarrow Q(s, a) + \mu(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
        ∗ set $s \leftarrow s'$
    – For each step of the current episode
• Until there are no more episodes
```

- The SARSA Algorithm



The Sarsa Algorithm

```
• Initialisation
    – set $Q(s,a)$ to small random values for all $s$ and $a$
• Repeat:
    – initialise $s$
    – choose action $a$ using the current policy
    – repeat:
        ∗ take action $a$ and receive reward $r$
        ∗ sample new state $s'$
        ∗ choose action $a'$ using the current policy
        ∗ update $Q(s,a) \leftarrow Q(s,a) + \mu(r + \gamma Q(s',a') - Q(s,a))$
        ∗ $s \leftarrow s'$, $a \leftarrow a'$
    – for each step of the current episode
• Until there are no more episodes
```

On-policy vs off-policy learning

- Reward structure:
  - Each move: -1
  - Move to cliff: -100
- Policy:
  - 90% chance of choosing best action (exploit). 10% chance of choosing random action (explore)

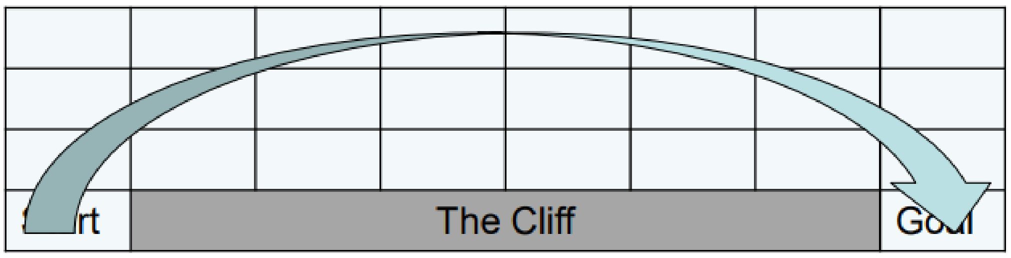
Q-learning

- Always assumes optimal action -> does not visit cliff often while learning. Therefore, does not learn that cliff is dangerous
- Resulting path is efficient, but risky

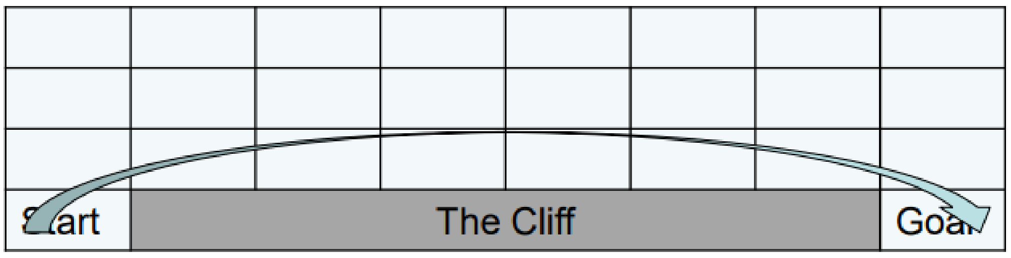
SARSA

- During learning, we more frequently end up outside the cliff (due to 10% chance of exploring in our policy)
- That info propagates to all states, generating a safer plan

- SARSA (on-policy):



- Q-learning (off-policy):



Conclusion from weekly exercise 12

Two core RL problems:

- Policy evaluation
- Policy improvement

The problem with assessing how good the policy is.

For **deterministic policies** we can solve the problem simply by computing the *total discounted return* of a policy. There is an inherent limitation to deterministic policies, so a solution might be **stochastic policies**, and we compute their *return in expectation* relying on *Monte Carlo methods.*

For policy improvement, the problem is learning optimal policies. A global value for a policy is not handy for editing the policy. Instead local values would be more useful. We can then compute **state values** and **action values** to assess a policy step by step, and use these values to choose improved actions. This iterative approach, based on evaluating the policy and improving the policy, is an instance of the more generic method **Genralized Policy Iteration**.

We can also consider a more advanced RL algorithm: Q-Learning.

Q-Learning allows us to leartn a optimal policy within an environment; this algorithm has two important features: (1) it learns in a continuous way, step-by-step, and not on complete episodes. (2) It learns an optimal policy while behaving according to a different, more exploration-prone policy.