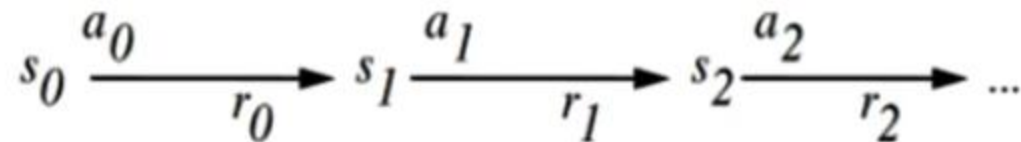
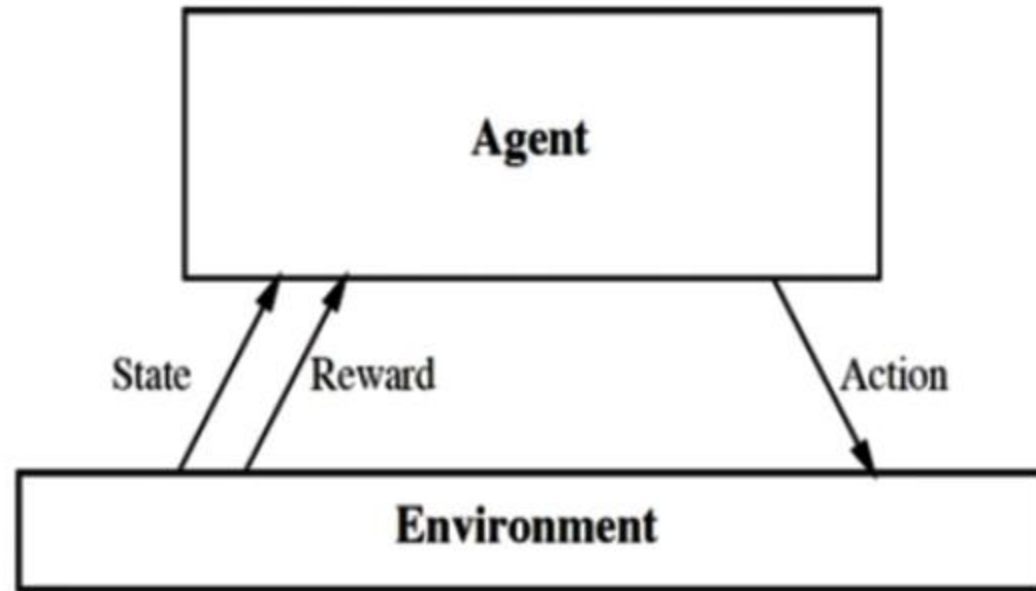


# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Reinforcement Learning



# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Reinforcement Learning

The intuitive scenario for Reinforcement Learning is an **Agent** that learns from interaction with an **Environment** to achieve some long-term goals related to the **State** of the environment by performing a sequence of actions and by receiving feedback.

The mapping from states to possible actions is called a **Policy**.

The achievement of goals is defined by **Rewards** or reward signals being the feedback on actions from the environment.

The **Return** is defined as the cumulative (possibly **discounted**) rewards over an **Episode** = action sequence, leading to a **Terminal state**.

The goal of any RL algorithm is to establish a policy that maximizes the **Returns**.

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

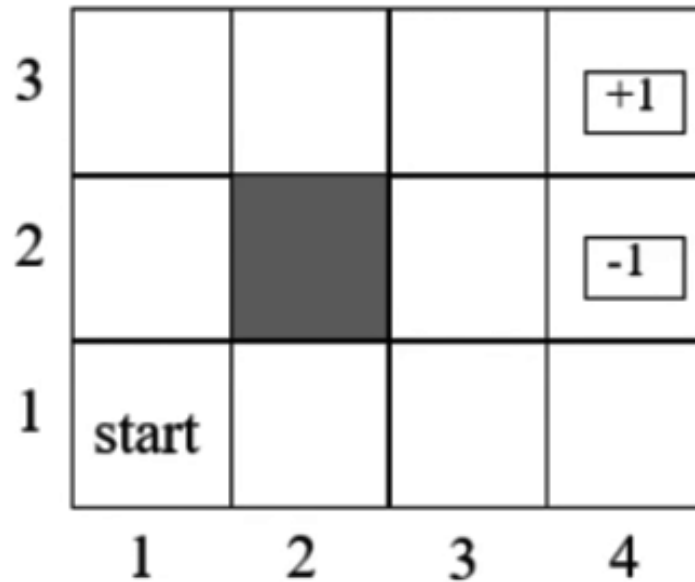
### Reinforcement Learning Terminology

<b>Environment</b>	The 'microworld' defined for the particular RL problem including the agent. Designated as E.
<b>Agent</b>	An agent Designated as A.
<b>State</b>	A particular configuration of the agent within the environment. Designated as s.
<b>Terminal states</b>	Defined end states for the particular RL problem. Designated as TS.
<b>Action</b>	An agent select an action based upon the current state S and the policy $\pi$
<b>Policy</b>	A policy $\pi$ is a mapping from states of the environment to the potential actions of an agent in those states. $\pi(s)$ can be deterministic, only depending on s or stochastic $\pi(s,a)$ depending also on a.
<b>Transition Probability function</b>	$s' = T(s, a)$ specifies the probability that the environment will transition to state $s'$ if the agent takes action a in state s.
<b>Episode</b>	Sometimes called an epoque. A sequence of states, actions and rewards, which ends in a terminal state.
<b>Reward or Reward signal</b>	The reward $R(s' s,a)$ gives feedback from the environment on the effect of a single action a in state s leading to $s'$
<b>Discounted Reward</b>	When calculating the Return, the expected rewards for future steps can be weighted with a discount factor $\gamma$ in the interval [0,1]
<b>Return</b>	The accumulated rewards for an episode. Designated as G.
<b>Value function</b>	The Value function $V = V(s)$ is the estimation of the Value or Utility of s with respect to its average Return considering all possible episodes possible within the current policy. It must continually be re-estimated for each action taken. The state-value function $v(s)$ for the policy $\pi$ is given below. Note that the value of the terminal state (if any) is always zero.
<b>Model of the environment</b>	A set of tuples $T(s,a)=s'$ , and $R(s' s,a)$ representing the possible state transitions and the rewards given.

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Reinforcement Learning: Example



# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Reinforcement Learning: Example

**Environment and agent:** The 4x3 world is a board of  $4 \times 3$  positions, indexed by two coordinates. The agent has a start position of (1,1). The position (2,2) is excluded.

**Reward:** There are two terminal positions (4,3) and (4,2). Reaching (4,3) gives a reward  $=+1$ . Reaching (4,2) gives a reward  $= -1$ . Reaching all other positions give reward  $=0$ .

**Actions:** up, down, left, right but restricted by the board configuration. The policy is deterministic in the sense that an action in a specific state can only lead to one other State.

**Episode:** e.g a sequence of actions leading from 1,1 to 4,3 via 1,2 1,3 2,3 and 3,3

Examples of episodes with returns.

$(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (3,4) \text{ } \underline{+1}$

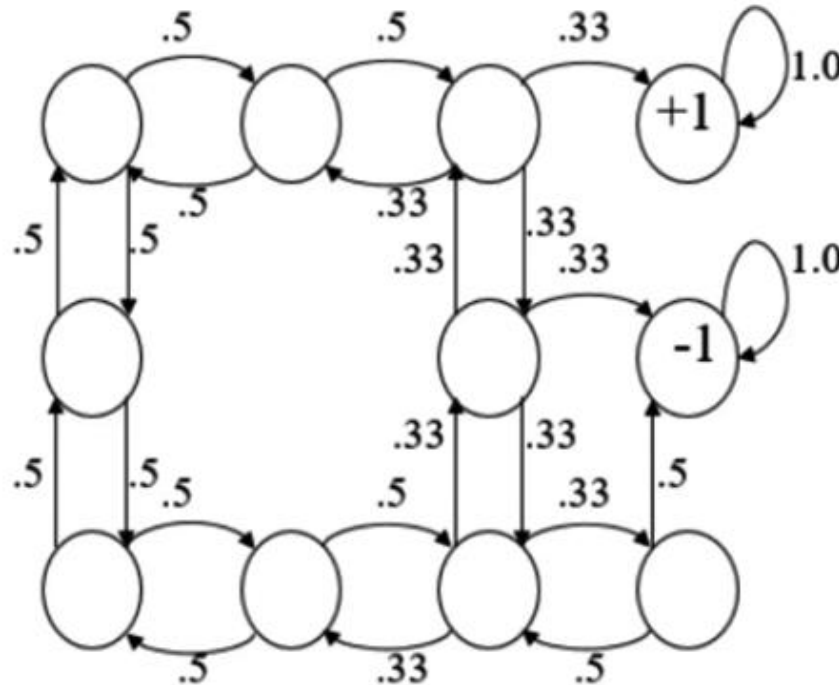
$(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (3,2) \rightarrow (3,3) \rightarrow (3,4) \text{ } \underline{+1}$

$(1,1) \rightarrow (2,1) \rightarrow (3,1) \rightarrow (3,2) \rightarrow (4,2) \text{ } \underline{-1}$

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Reinforcement Learning as Markov Decision Process (MDP)



**State transitions:** The state transitions from a state to its neighboring state that in this case have equal probability, summed up to 1.



# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Reinforcement Learning as Markov Decision Process (MDP)

A suitable model for the above intuitive scenario is a basic **Markov Decision Process (MDP)**.

An MDP is typically defined by a 4-tuple  $(S, A, R, T)$  where:

- $S$  is the set of states  $s$  is the state of the environment (including the agent)
- $A$  is the set of actions for each  $s$ , that the agent can choose between defined by a policy  $\pi$
- $R(s' | s, a)$  is a function that returns the reward received for taking action  $a$  in state  $s$  leading to  $s'$
- $s' = T(s, a)$  is a transition probability function, specifying the probability that the environment will transition to state  $s'$  if the agent takes action  $a$  in state  $s$ . The *Markov property* = Transition probabilities depend on state only, not on the path to the state.

The goal is to find a policy  $\pi$  that maximizes the return = expected future accumulated (discounted) reward for episodes from a Start state  $s$  to a Terminal state.

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Reinforcement Learning vs Reinforcement Learning

In one MDP scenario, there is a complete and exact model of the MDP in the sense that  $T(S,A)$  and  $R(S,A)$  are fully defined.

**In this case a MDP problem is a Planning Problem that can be exactly solved by use of e.g. Dynamic Programming.**

However in many cases  $T(S,A)$  and  $R(S,A)$  are not completely known, meaning that the model of the MDP is not complete.

**In such cases we have a true Reinforcement Learning problem.**



# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Dynamic Programming

Dynamic Programming is an algorithm design technique for optimization problems: often for minimizing or maximizing. The method was developed by Richard Bellman in the 1950s.

Like divide and conquer, DP is simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner and then combining the solutions to the sub-problems to form the total solution.

If a problem can be optimally solved by breaking it recursively into sub-problems and then form the solution from optimal solutions to the sub-problems, then it is said to have optimal substructure.

Unlike divide and conquer, sub-problems are not independent, sub-problems may share sub-sub-problems,

Typically the utility value of a solution in a certain state is defined by a value function calculated recursively based on the value functions for the remaining steps of an episode moderated by the relevant probabilities and a discount parameter giving higher weight to closer reward contributions. Typically the policy function can be calculated in a similar fashion.

The defining equation for finding an optimal value function is called the Bellman equation.

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Bellman's principle's of optimality and the Bellman equation

Dynamic programming algorithms has the ambition to optimize the following two functions:

**Policy function:**  $\pi(s) := \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a) * (R(s'|s,a) + \lambda * V(s'))$

**Value function:**  $V(s) := \sum_{s'} T(s,a) * (R(s'|s,a) + \lambda * V(s'))$

#### Richard Bellman's Principle of Optimality

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

**An optimal value function** is a solution to the so called **Bellman equation**:

$$V(s) = \operatorname{Max}_{i=1..n} ( R(s'|s,ai) + \lambda * V(s' = T(s,ai)) )$$

**An optimal value function can be the basis for an optimal policy function.**

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Standard approaches to solve the Bellman equation

#### Value iteration

Initialise  $V$  arbitrarily

repeat

    Improve  $V_{k+1}$  using the estimate of  $V_k$

until convergence

#### Policy iteration

Initialise  $V$  and  $\pi$  arbitrarily

repeat

    Evaluate  $V$  using  $\pi$

    Improve  $\pi$  using  $V$

until convergence.

#### Linear programming

**Prioritized sweeping** – use priority queue to update states with large potential for change

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Distinctions in Reinforcement Learning

- Passive versus Active learning
- On-policy versus Off-policy learning
- Exploitation versus Exploration learning
- Model-based versus Model-free learning

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Passive Learning vs Active Learning

#### **Passive learning**

A Passive Agent executes a fixed policy and evaluates it. The agent simply watches the world going by and tries to learn the utilities of being in various states.

#### **Active learning**

An Active agent updates its policy as it learns while acting in an uncertain world. An active agent must consider what actions to take, what their outcomes may be, and how they affect the rewards achieved via exploration. Active learning is most typical for the true reinforcement learning cases.



# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### On-Policy vs Off-Policy

In the Planning DP case an agent could conduct **off-policy planning**, that is, formulate a policy without needing to interact with the world. In **off-policy** methods, the agent's policy used to choose its actions is called the *behavior* policy which may be unrelated to the policy that is evaluated and improved, called the *estimation* policy.

In the situation that a complete model of the environment is NOT available for the agent, i.e. elements of  $T(s,a)$  and  $R(s'|s,a)$  are lacking, and then we must typically engage in **on-policy planning**, in which we must interact with the world to better estimate the value of a policy while using it for control. In the on-policy mode the planning and learning parts are interwoven.



# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Exploration vs Exploitation

**Actions in RL systems can be of the following two kinds:**

**Exploitation actions:** have a preference for past actions that have been found to be effective at producing reward and thereby exploiting what is already known. The Planning DP case is completely in this genre.

**Exploration actions:** have a preference for exploring untested actions to discover new and potentially more reward-producing actions. Exploration is typical for the true reinforcement learning cases.

Managing the trade-off between exploration and exploitation in its policies is a critical issue in RL algorithms.

- One guideline could be to explore more when knowledge is weak and exploit more when we have gained more knowledge.
- One method:  $\epsilon$ -greedy keep to exploitation as long as actions has the probability  $1-\epsilon$ . If no action which satisfies this condition is found, the agent turns to exploration.  $\epsilon$  is a hyper-parameter controlling the balance between exploitation and exploration.

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Model-free vs Model-based Reinforcement Learning

In the situation that a complete model of the environment is NOT available for the agent, i.e. elements of  $T(s,a)$  and  $R(s'|s,a)$  are lacking, RL offers two different approaches.

#### Model-based RL algorithms.

One approach is to try to learn an adequate model of the environment and then fall back on the planning task for a complete model to find a policy.

That is, if the agent is currently in state  $s$ , takes action  $a$ , and then observes the environment transition to state  $s'$  with reward  $r$ , that observation can be used to improve its estimate of  $T(s, a)$  and  $R(s'|s,a)$  through supervised learning techniques.

#### Model-free algorithms.

However a model of the environment is not necessary to find a good policy.

One of the most classic examples is *Q-learning*, which directly estimates optimal so called *Q*-values of each action in each state (related to the utility of each action in each state), from which a policy may be derived by choosing the action with the highest *Q*-value in the current state.

# **Introduction to Machine Learning**

## **Semi Supervised and Reinforcement Learning**

### **Solving a Reinforcement Learning Problem**

- **Model-based approaches**
  - a. **Adaptive Dynamic Programming**
  
- **Model-Free Approaches**
  - a. **Direct Estimate**
  - b. **Monte Carlo Simulation**
  - c. **Temporal-Difference Learning**
    - Q-Learning- off policy**
    - SARSA- on-policy**

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Model free Direct Estimation of the value function

Estimate the value function for a particular state  $s$  as the average of total rewards of a sample of epochs (calculating from  $s$  to end of epoch)

The so called 'Reward to go' of a state  $s$  is the sum of the (discounted) rewards from that state until a terminal state is reached. The estimated value of the state is based on the 'Rewards to go'.

Direct Utility Estimation keeps a running average of the observed 'rewards-to-go' for each state  $s$ .

Value ( $s$ ) = average of ( all ( 'Reward to go' ( $s$ ) for all episodes  $e$  in the sample)

As the number of trials goes to infinity, the sample average converges to the true utility for state  $s$ .


Drawback: very slow convergence.

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Model based Adaptive Dynamic Programming

The strategy in ADP is to first complete the partially known MDP model and then regard it as complete applying the Dynamic Programming technique as in the planning case with complete knowledge.

$$V(s) := \sum_{s'} T(s,a) * (R(s'|s,a) + \gamma * V(s'))$$


To be learned

Algorithm for iteratively updating estimates of  $R(s'|s,a)$  and  $T(s,a)$

1. Collecting examples of rewards for  $s, a, s$  triples
2. Collecting examples of  $s, a$  and  $s'$  triples
3. Taking averages of collected rewards
4. Calculate the fraction of time  $s,a$  leads to  $s'$ .

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Model free Learning: Monte Carlo Simulation

Monte Carlo methods are computational algorithms that are based on Repeated random samples to estimate some target properties of a model.

Monte Carlo Simulations simply apply Monte Carlo methods in the context of Simulations.

Monte Carlo Reinforcement Learning MC methods learn directly from samples of complete episodes of experience.

MC is model-free: no explicit model of MDP transitions and rewards is built up.

MC takes mean returns for states in sampled episodes. The value expectation of a state is set to the iterative mean of all the empirical returns (not expected) for the episodes.

First-visit MC: average returns only for first time  $s$  is visited in an episode.

Every-Visit MC: average returns for every time  $s$  is visited in an episode.



# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Algorithm First-Visit Monte Carlo

1. Initialize the policy, state-value function arbitrarily 2: Returns(s)  $\leftarrow$  empty list
2. Start by generating an episode E according to the current policy  $\pi$ .
3. For each s in E  
    Begin  
        4. if this is the first occurrence of this state add the return received to the returns list.  
        5. calculate an iterative means (average) for all returns in the list  
        6. set the value of the state as the computed average  
    End  
Repeat step 2-4 until convergence.

The every-visit algorithm simply modifies step 4.

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Using an Incremental Mean in Monte Carlo Simulation

It is convenient to convert the mean return into an incremental update so that the mean can be updated with each episode and one can follow the progress made with each episode.

$V(s)$  is incrementally updated for each state  $S_t$ , with return  $G_t$ .

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)} (G_t - V(S_t))$$

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Model-free Learning: Temporal difference learning

**Temporal difference (TD) learning** is a class of model-free reinforcement learning methods which learn by bootstrapping from the current estimate of the value function.

TD algorithms:

- sample from the environment, like Monte Carlo simulations and
- perform updates based on current estimates like adaptive dynamic programming methods.

While Monte Carlo methods only adjust their estimates once the final outcome is known, TD methods adjust predictions before the final outcome is known. Typically TD algorithms adjust the estimated utility value of the current state  $s$  based on its immediate reward and the estimated value of the next state  $s'$ . The term 'temporal' is motivated by the temporal relation between states  $s$  and  $s'$ .

The Temporal difference equation

$$V(s) \leftarrow V(s) + \alpha * (R(s,a) + \gamma * V(s') - V(s))$$

where  $\alpha$  is a learning rate parameter (hyper parameter).

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Q Learning

**Q-learning** is a model-free off-policy TD reinforcement learning algorithm.

The goal of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances.

For any finite Markov decision process (FMDP), *Q*-learning finds a policy that is optimal in the sense that it maximizes the expected value of the total reward over any and all successive steps, starting from the current state.

*Q*-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy.

"Q" names the function  $Q(s, a)$  that can be said to stand for the "quality" of an action  $a$  taken in a given state  $s$ .

Suppose we have the optimal *Q*-function  $(s, a)$  then the optimal policy in state  $s$  is  $\text{argmax}_a Q(s, a)$ .

# Introduction to Machine Learning

## Semi Supervised and Reinforcement Learning

### Q Learning Algorithm

Initialize  $Q(s, a)$  arbitrarily

Repeat (for each episode)

Initialize  $s$

Repeat (for each step of the episode)

Take action  $a$ , observe  $r, s'$

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

$$s \leftarrow s'$$

*With  $\alpha = 1$  or  $\alpha = 1$  and  $\gamma = 1$  the updating formula is simplified*

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

$$Q(s, a) \leftarrow r + \max_{a'} Q(s', a')$$