Def. AI:

Artificial intelligence is that activity devoted to making machines intelligent, and intelligence is that quality that enables an entity to function appropriately and with foresight in its environment.

Def AI Effect:

The great practical benefits of AI applications and even the existence of AI in many software products go largely unnoticed by many despite the already widespread use of AI techniques in software. This is the AI effect.

Example: A modern calculator is assumed to have no AI whereas when its previous version, Arithometer was discovered it was said to have AI in doing calculations.

Machine Learning:

Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that can change when exposed to new data.

Deep Learning:

Deep Learning is classified as a class of machine learning algorithm that use a cascade of many layers of nonlinear processing units for feature extraction and transformation. It is based on the (unsupervised) learning of multiple levels of features or representations of the data. Higher level features are derived from lower level features to form a hierarchical representation.

Agents:

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

Rationality:

A rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Chinese room experiment:

The Chinese room argument holds that a program cannot give a computer a "mind", "understanding" or "consciousness", regardless of how intelligently or human-like the program may make the computer behave.

Suppose that artificial intelligence research has succeeded in constructing a computer that comfortably passes the Turing test: it convinces a human Chinese speaker that the program is itself a live Chinese speaker.

The question Searle wants to answer is this: does the machine literally "understand" Chinese? Or is it merely simulating the ability to understand Chinese?

Searle can sit in a room and perform the program instructions step-by-step manually. However, Searle would not be able to understand the conversation.

Types of agents:

Rational agent: A rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Omniscient agent: An omniscient agent knows the actual outcome of its actions and can act accordingly; but omniscience is impossible in reality.

Simple reflex agent: These agents select actions on the basis of the current percept, ignoring the rest of the percept history.

Model-based agent: current percept is combined with the old internal state to generate the updated description of the current state, based on the agent's model of how the world works. Knowledge about "how the world works"—whether implemented in simple Boolean circuits or in complete scientific theories—is called a model of the world.

Goal-based agent:

As well as a current state description, the agent needs some sort of goal information that describes situations that are desirable. Example: The reflex agent brakes when it sees brake lights. A goal-based agent, in principle, could reason that if the car in front has its brake lights on, it will slow down. Given the way the world usually evolves, the only action that will achieve the goal of not hitting other cars is to brake.

Utility-based agent:

An agent's utility function is essentially an internalization of the performance measure. If the internal utility function and the external performance measure are in agreement, then an agent that chooses actions to maximize its utility will be rational according to the external performance measure.

Learning agent:

The learning element uses feedback from the "critic" on how the agent is doing and determines how the performance element should be modified to do better in the future.

Properties of Environment:

Fully observable vs. partially observable: If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable. If the agent has no sensors at all then the environment is unobservable.

Single agent vs. multi-agent: example, an agent solving a crossword puzzle by itself is clearly in a single-agent environment, whereas an agent playing chess is in a two agent environment.

Deterministic vs. stochastic: If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is deterministic; otherwise, it is stochastic.

Episodic vs. sequential: In an episodic the next episode does not depend on the actions taken in previous episodes. In sequential environments, on the other hand, the current decision could affect all future decisions.

Static vs. dynamic: If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise, it is static.

Discrete vs. continuous: The discrete/continuous distinction applies to the state of the environment, to the way time is handled, and to the percepts and actions of the agent. Example: Chess also has a discrete set of percepts and actions. Taxi driving is a continuous-state and continuous-time problem.

Known vs. unknown: In a known environment, the outcomes for all actions are given. Obviously, if the environment is unknown, the agent will have to learn how it works in order to make good decisions.

Perceptron Learning Algorithm: read from slides classnotes-1

Figure 1 provides a useful visualisation of a Perceptron. For $d \geqslant 1$, a d-dimensional Perceptron has a d-dimensional parameter vector $\mathbf{w} \in \mathbb{R}^d$. For input $\mathbf{x} \in \mathbb{R}^d$, the output produced is $y = \text{sign}(\mathbf{w} \cdot \mathbf{x})$, where for $\alpha \in \mathbb{R}$,

$$\text{sign}(\alpha) = \begin{cases} 1 \text{ if } \alpha \geqslant 0, \\ 0 \text{ if } \alpha < 0. \end{cases}$$

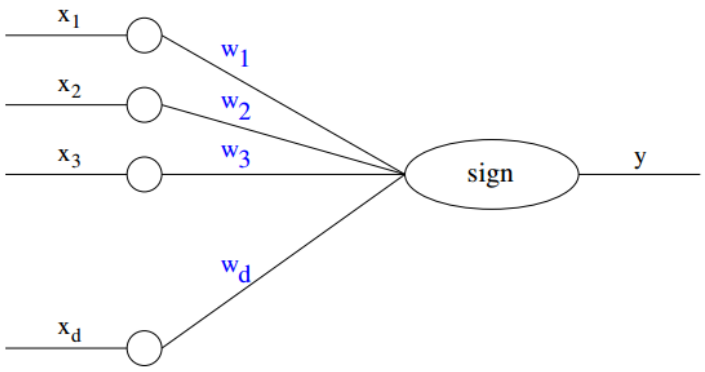


Figure 1: Perceptron

Input:

$$(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \ldots, (\mathbf{x}^n, y^n),$$

where for $i \in \{1, 2, \ldots, n\}$, $\mathbf{x}^i \in \mathbb{R}^d$ and $y^i \in \{-1, 1\}$.

Assumption 1 (Linear Separability). *There exists some* $\mathbf{w}^\star \in \mathbb{R}^d$ *such that* $||\mathbf{w}^\star|| = 1$ *and for some* $\gamma > 0$, *for all* $i \in \{1, 2, \ldots, n\}$,

$$y^i(\mathbf{w}^\star \cdot \mathbf{x}^i) > \gamma.$$

Assumption 2 (Bounded coordinates). *There exists* $R \in \mathbb{R}$ *such that for* $i \in \{1, 2, \ldots, n\}$,

$$||\mathbf{x}^i|| \leqslant R.$$

Perceptron Learning Algorithm

$k \leftarrow 1; \mathbf{w}_k \leftarrow \mathbf{0}$.
While there exists $i \in \{1, 2, \ldots, n\}$ such that $y^i(\mathbf{w}_k \cdot \mathbf{x}^i) \leqslant 0$:
    Pick an arbitrary $j \in \{1, 2, \ldots, n\}$ such that $y^j(\mathbf{w}_k \cdot \mathbf{x}^j) \leqslant 0$.
    $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + y^j x^j$.
    $k \leftarrow k + 1$.
Return $\mathbf{w}_k$.

Theorem 3 (Perceptron convergence). *The Perceptron Learning Algorithm makes at most* $\frac{R^2}{\gamma^2}$ *updates (after which it returns a separating hyperplane).*

Neural Networks:

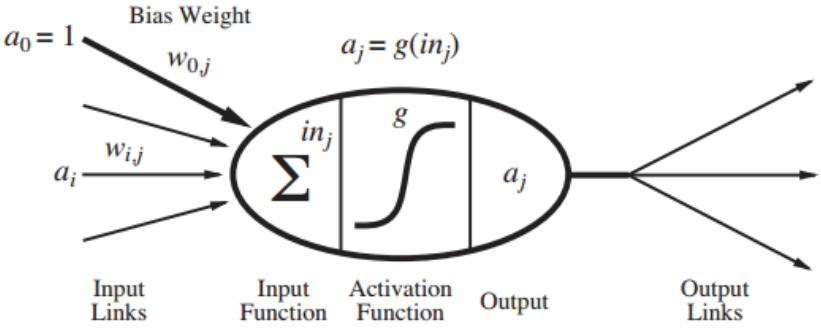


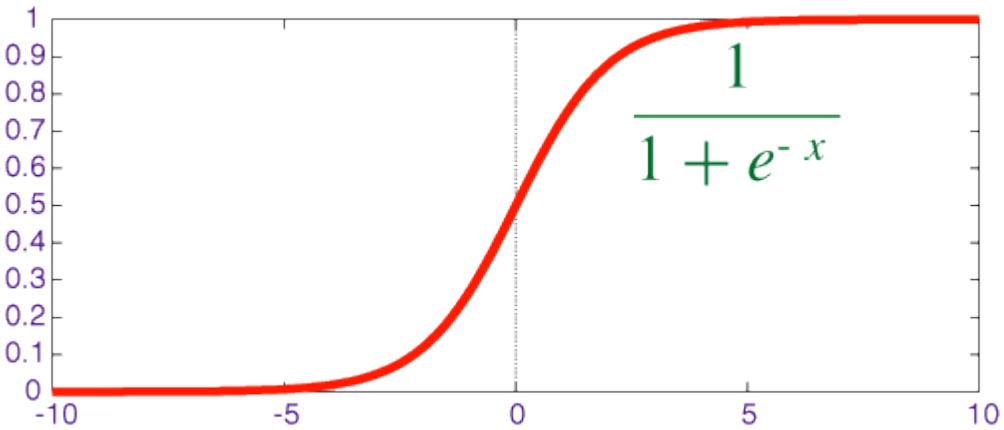
Figure 18.19 A simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^{n} w_{i,j} a_i)$, where a_i is the output activation of unit i and $w_{i,j}$ is the weight on the link from unit i to this unit.

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i$$

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right)$$

If the activation function g is hard then it is called **perceptron** else if a logistic function is used it is called as **sigmoid perceptron**.

Sigmoid activation function:



A **feed-forward network** has connections only in one direction—that is, it forms a directed acyclic graph.

A **recurrent network**, on the other hand, feeds its outputs back into its own inputs.

Perceptrons are single layer feed-forward network.

$$\frac{\partial}{\partial w} Loss(\mathbf{w}) = \frac{\partial}{\partial w} |\mathbf{y} - \mathbf{h_w}(\mathbf{x})|^2 = \frac{\partial}{\partial w} \sum_k (y_k - a_k)^2 = \sum_k \frac{\partial}{\partial w} (y_k - a_k)^2$$

$$\Delta_k = Err_k \times g'(in_k).$$

$$w_{j,k} \leftarrow w_{j,k} + \alpha \times a_j \times \Delta_k$$

$$\Delta_j = g'(in_j) \sum_k w_{j,k} \Delta_k$$

function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network
inputs: *examples*, a set of examples, each with input vector **x** and output vector **y**
network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
    /* *Propagate the inputs forward to compute the outputs* */
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $\ell = 2$ **to** L **do**
      for each node j in layer ℓ **do**
        $in_j \leftarrow \sum_i w_{i,j} a_i$
        $a_j \leftarrow g(in_j)$
    /* *Propagate deltas backward from output layer to input layer* */
    for each node j in the output layer **do**
      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
    for $\ell = L - 1$ **to** 1 **do**
      for each node i in layer ℓ **do**
        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$
    /* *Update every weight in network using deltas* */
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

Figure 18.24 The back-propagation algorithm for learning in multilayer networks.

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j$$

Supervised Learning:

The task of supervised learning is this:

Given a **training set** of N example input–output pairs

$$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N) ,$$

where each y_j was generated by an unknown function $y = f(x)$, discover a function h that approximates the true function f.

Decision Trees:

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Figure 18.3 Examples for the restaurant domain.

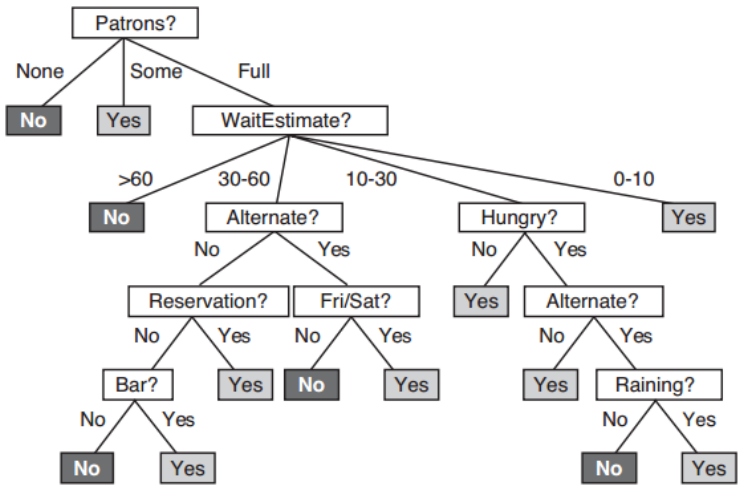


Figure 18.2 A decision tree for deciding whether to wait for a table.

Gradient Descent:

$\mathbf{w} \leftarrow$ any point in the parameter space

loop until convergence **do**

  for each w_i **in w do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

alpha is called learning factor.

Cross-validation (k-fold cross validation):

In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k − 1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged to produce a single estimation.

Advantage: all observations are used for both training and validation, and each observation is used for validation exactly once.

Nearest-neighbor method of classification:

The class(y) of a query point is decided by taking average of its k-nearest neighbors.

Distance here can be calculated here using various methods. (E.g. Euclidean distance)

K-mean clustering Problem: Read from classnote-2

k-means Clustering Algorithm

Let $\mathcal{C}^0$ be an arbitrary clustering, and let $\boldsymbol{\mu}^0 = (\boldsymbol{\mu}^1, \boldsymbol{\mu}^2, \ldots, \boldsymbol{\mu}^k)$ be a sequence of centres such that for $k' \in \{1, 2, \ldots, k\}$, $\boldsymbol{\mu}^0_{k'}$ is the centroid of the points in the k'-th cluster.
$t \leftarrow 0$.
converged $\leftarrow$ false.
While $\neg$converged
    converged $\leftarrow$ true.
    for $i \in \{1, 2, \ldots, n\}$
        $\mathcal{C}^{t+1}(i) \leftarrow \mathcal{C}^t(i)$.
        for $k' \in \{1, 2, \ldots, k\}$
            If $k' \neq \mathcal{C}^{t+1}(i)$ and $\|\boldsymbol{x}^i - \boldsymbol{\mu}_{k'}\| < \|\boldsymbol{x}^i - \boldsymbol{\mu}_{\mathcal{C}^{t+1}(i)}\|$
                $\mathcal{C}^{t+1}(i) \leftarrow k'$.
                converged $\leftarrow$ false.
    for $k' \in \{1, 2, \ldots, k\}$
        Set $\boldsymbol{\mu}^{t+1}_{k'}$ to be the centroid of all points i such that $\mathcal{C}^{t+1}(i) = k'$.
    $t \leftarrow t + 1$.
Return $\mathcal{C}^t, \boldsymbol{\mu}^t$.

Lemma 1. *Consider the points $\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^m$, where $m \geq 1$, and for $i \in \{1, 2, \ldots, m\}$, $\boldsymbol{z}^i \in \mathbb{R}^d$. Let $\bar{\boldsymbol{z}} = \frac{1}{m}\sum_{i=1}^m \boldsymbol{z}^i$ be the mean of these points, and let $\boldsymbol{z} \in \mathbb{R}^d$ be an arbitrary point in the same (d-dimensional) space. Then*

$$\sum_{i=1}^m \|\boldsymbol{z}^i - \boldsymbol{z}\|^2 \geq \sum_{i=1}^m \|\boldsymbol{z}^i - \bar{\boldsymbol{z}}\|^2.$$

Theorem 2. *The k-means clustering algorithm converges.*

We show that $\text{SSE}(\mathcal{C}^{t+1}, \boldsymbol{\mu}^{t+1}) < \text{SSE}(\mathcal{C}^t, \boldsymbol{\mu}^t)$ in two steps. First, we show that

$$\text{SSE}(\mathcal{C}^{t+1}, \boldsymbol{\mu}^t) < \text{SSE}(\mathcal{C}^t, \boldsymbol{\mu}^t), \tag{1}$$

and next, we show that

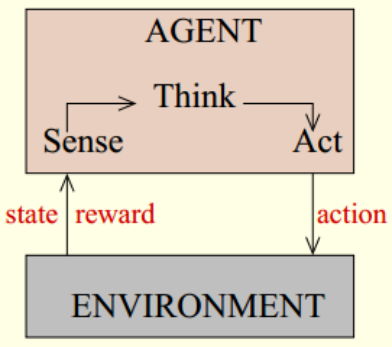
$$\text{SSE}(\mathcal{C}^{t+1}, \boldsymbol{\mu}^{t+1}) \leq \text{SSE}(\mathcal{C}^{t+1}, \boldsymbol{\mu}^t). \tag{2}$$

$$\text{SSE}(\mathcal{C}^{t+1}, \boldsymbol{\mu}^t) = \sum_{i=1}^n \|\boldsymbol{x}^i - \boldsymbol{\mu}^t_{C^{t+1}(i)}\|^2 < \sum_{i=1}^n \|\boldsymbol{x}^i - \boldsymbol{\mu}^t_{C^t(i)}\|^2 = \text{SSE}(\mathcal{C}^t, \boldsymbol{\mu}^t).$$

The second step puts Lemma 1 to use:

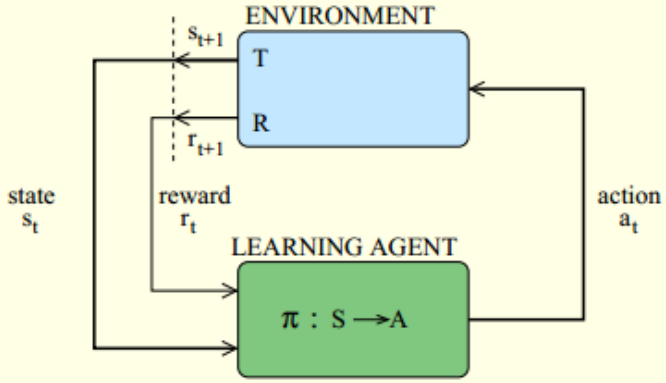
$$\begin{aligned}
\text{SSE}(\mathcal{C}^{t+1}, \boldsymbol{\mu}^{t+1}) &= \sum_{i=1}^n \|\boldsymbol{x}^i - \boldsymbol{\mu}^{t+1}_{C^{t+1}(i)}\|^2 \\
&= \sum_{k'=1}^k \sum_{i \in \{1,2,\ldots,n\}, \mathcal{C}^{t+1}(i)=k'} \|\boldsymbol{x}^i - \boldsymbol{\mu}^{t+1}_{C^{t+1}(i)}\|^2 \\
&\leq \sum_{k'=1}^k \sum_{i \in \{1,2,\ldots,n\}, \mathcal{C}^{t+1}(i)=k'} \|\boldsymbol{x}^i - \boldsymbol{\mu}^{t}_{C^{t+1}(i)}\|^2 \\
&= \sum_{i=1}^n \|\boldsymbol{x}^i - \boldsymbol{\mu}^t_{C^{t+1}(i)}\|^2 \\
&= \text{SSE}(\mathcal{C}^{t+1}, \boldsymbol{\mu}^t).
\end{aligned}$$

Reinforcement Learning:



Sequential Decision making situations: situations in which a decision is made, an event occurs, another decision is made, another event occurs, and so on.

Markov's Decision problem:



S: set of states.
A: set of actions.
T: transition function. $\forall s \in S, \forall a \in A$, $T(s, a)$ is a distribution over S.
R: reward function. $\forall s, s' \in S, \forall a \in A$, $R(s, a, s')$ is a finite real number.
γ: discount factor. $0 \leq \gamma < 1$.

Trajectory over time: $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_t, a_t, r_{t+1}, s_{t+1}, \ldots$.
Value, or expected long-term reward, of state s under policy π:

$$V^{\pi}(s) = \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots \text{ to } \infty | s_0 = s, a_i = \pi(s_i)].$$

Objective: "Find π such that $V^{\pi}(s)$ is maximal $\forall s \in S$."

Bellman's Equation:

$$V^{\pi}(s) = \sum_{s' \in S} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V^{\pi}(s') \right].$$

V^{π} is called the value function of π.

Define $(\forall s \in S, \forall a \in A)$:

$$Q^{\pi}(s, a) = \sum_{s' \in S} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi}(s') \right].$$

Q^{π} is called the action value function of π.

$V^{\pi}(s) = Q^{\pi}(s, \pi(s))$.

Bellman's Optimality Equations:

$$V^{*}(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s') \left[R(s, a, s') + \gamma V^{*}(s') \right].$$

Value Iteration method to solve Bellman's Optimality Equations: (planning problem, computationally efficient)

- Initialise $V^0 : S \to \mathbb{R}$ arbitrarily.
- $t \leftarrow 0$.
- Repeat
  - For all $s \in S$,
    - $V^{t+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s') \left[R(s, a, s') + \gamma V^{t}(s) \right]$.
  - $t \leftarrow t + 1$.
- Until $\| V^t - V^{t-1} \|$ is small enough.

Q-learning: a temporal difference learning algorithm: (Learning problem, sample efficient)

- Let Q be our "guess" of Q^*: for every state s and action a, initialise $Q(s, a)$ arbitrarily. We will start in some state s_0.
- For $t = 0, 1, 2, \ldots$
  - Take an action a_t, chosen uniformly at random with probability ϵ, and to be $\text{argmax}_a \, Q(s_t, a)$ with probability $1 - \epsilon$.
  - The environment will generate next state s_{t+1} and reward r_{t+1}.
  - Update: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t (r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t))$.
  - [ϵ: parameter for "ϵ-greedy" exploration] [α_t: learning rate]
  - [$r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)$: temporal difference prediction error]

Challenges: presence of state aliasing and generalisation