Artificial Intelligence
Learning in Agent Systems

More about

Textbook, Chapters on Learning

What is learning needed for?

- **Increasing intelligence**
  - Learning new task performing capacity
  - Recognizing affordances
  - Recognizing resources
- **Increasing usefull life-span by adaptivity**
  - Compensating for lack of knowledge in design
    - Learning (exploring) the environment
    - Learning the properties of other agents
  - Compensating for changes in the environment
- **Increasing robustness and fault-tolerant capability**
  - Learning self (percepts/actions) upon contingency

Learning agent

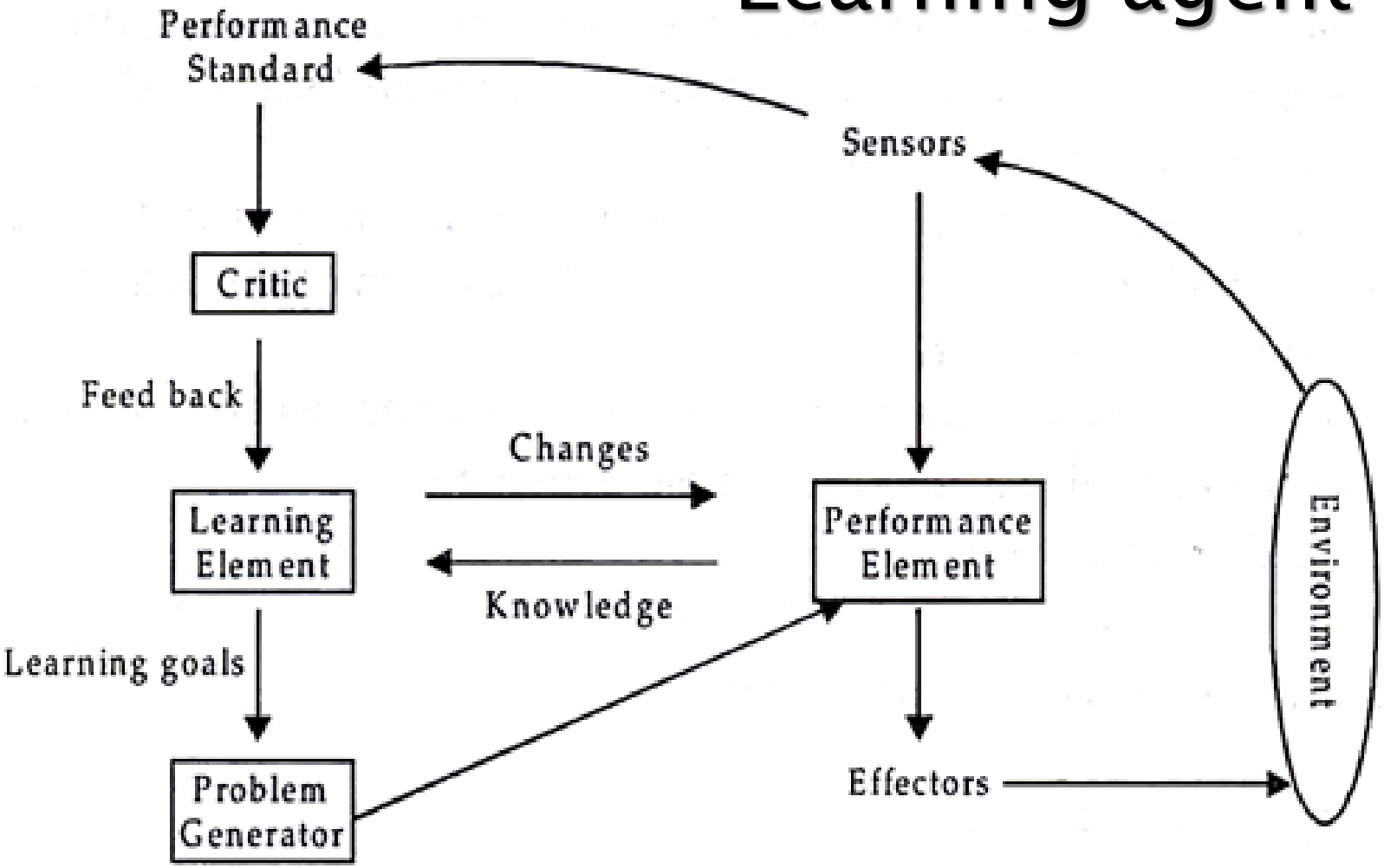


Fig. 9.1. *Model of learning agent.*

Abstract paradigm of function learning – Inductive learning

Learn a function from examples in some representation

f	is the target function
a pair (*x*, *f(x)*)	is an example

Problem:

find a hypothesis *h(x)*, such that $h(x) \approx f(x)$
given a training set of examples
and $h(x) \approx f(x)$, given a test set of examples
(generalization, abstraction)

(*h* is consistent if it agrees with *f* on all examples)

Learning agent

a priori knowledge

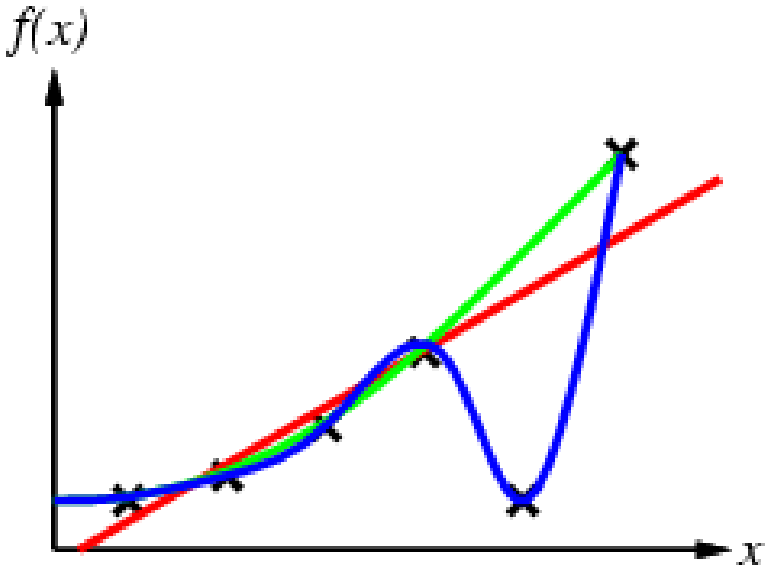
feedback supervised learning
reinforcement learning
unsupervised learning

learning = search in hypothesis space
learning bias
learning noise

expressiveness vs. efficiency

dynamics of learning vs. dynamics of environment

Inductive learning



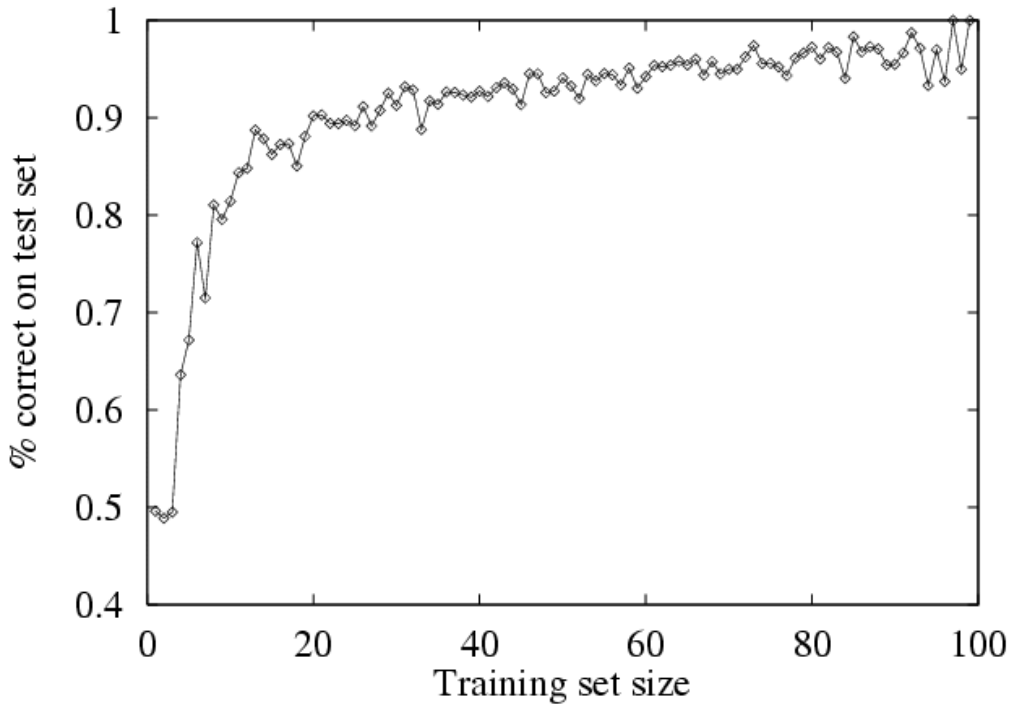
Learning bias

Learning noise

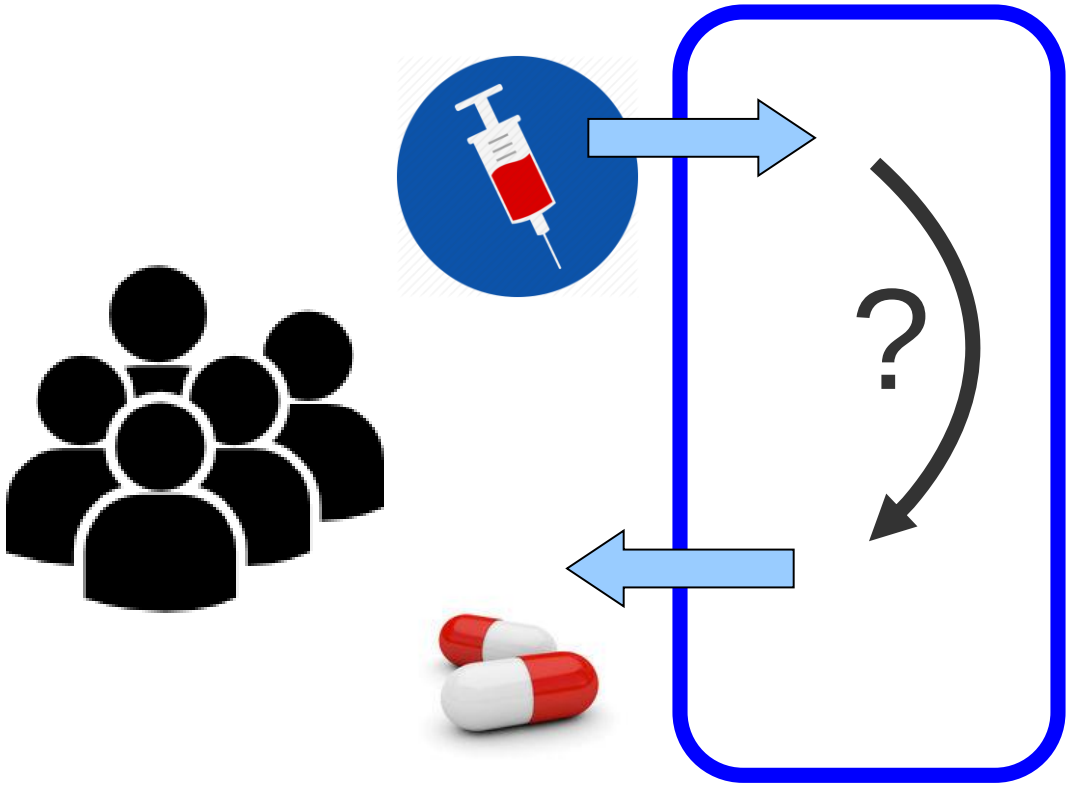
Estimation of future performance

How do we know that $h(x) \approx f(x)$?

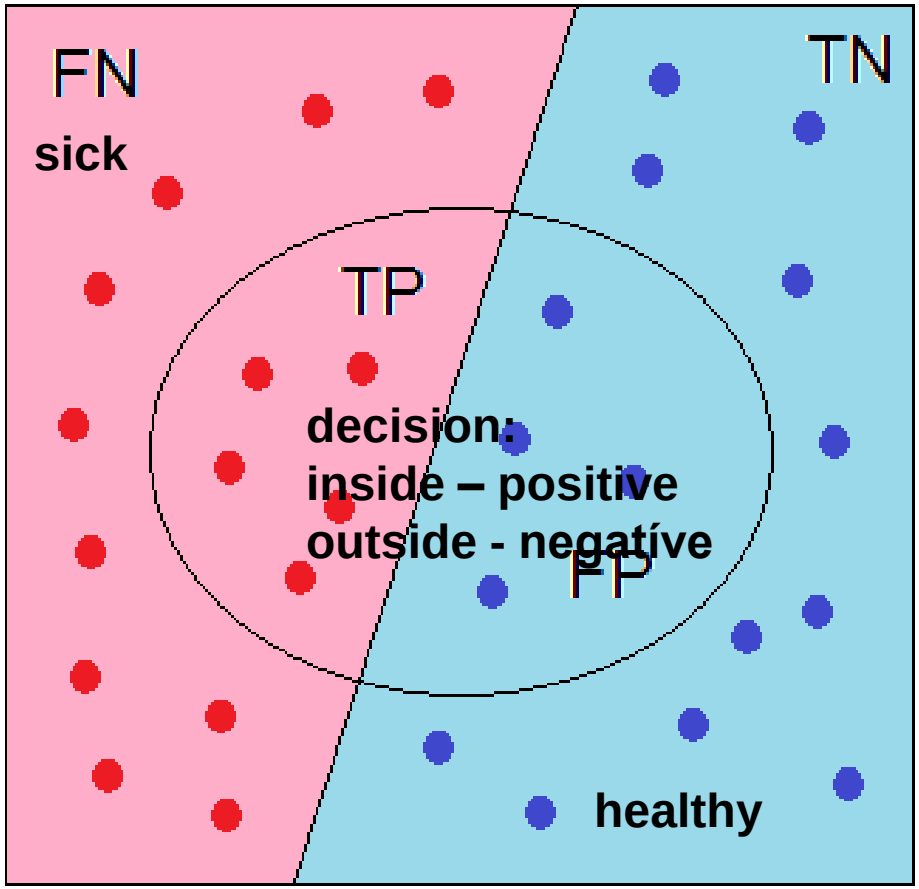
Learning curve =
% correct on test set as
a function of training set size



Binary classification = binary decision

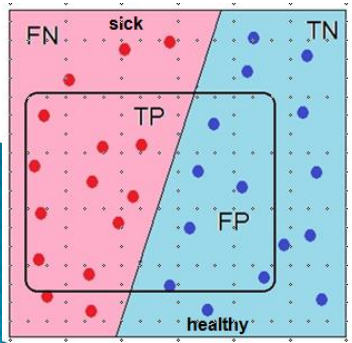


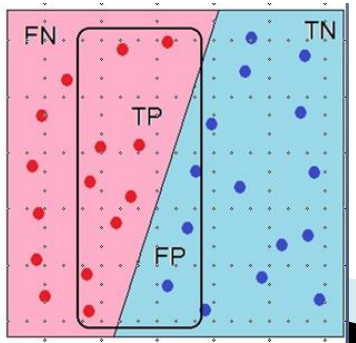
Binary classification = binary decision



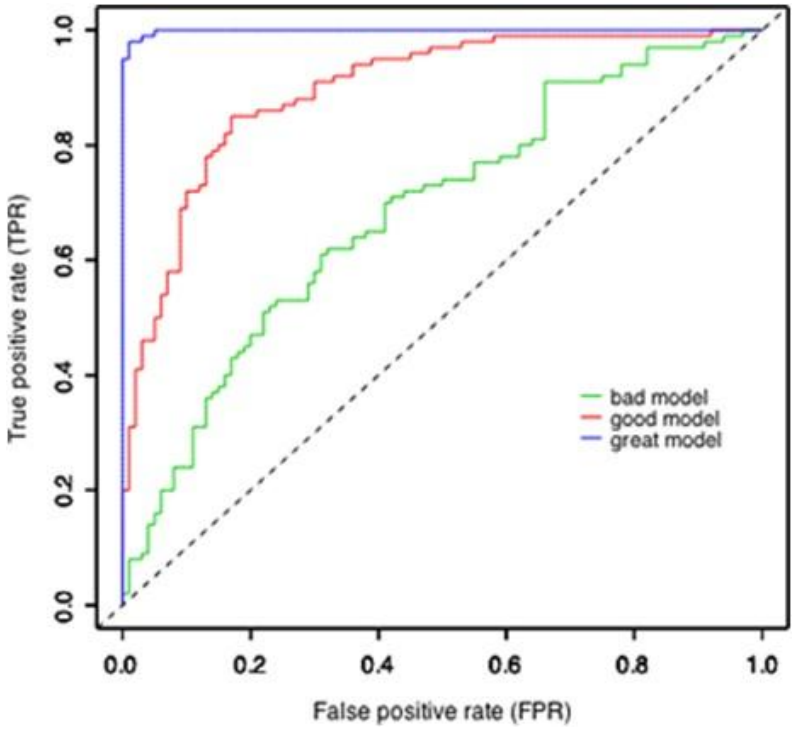
true positive rate (TPR)
TPR = TP/P = TP / (TP+FN)

false positive rate (FPR)
FPR = FP/N = FP / (FP+TN)





Which one is better?



Learning decision trees

Problem:

decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Attribute-based representations

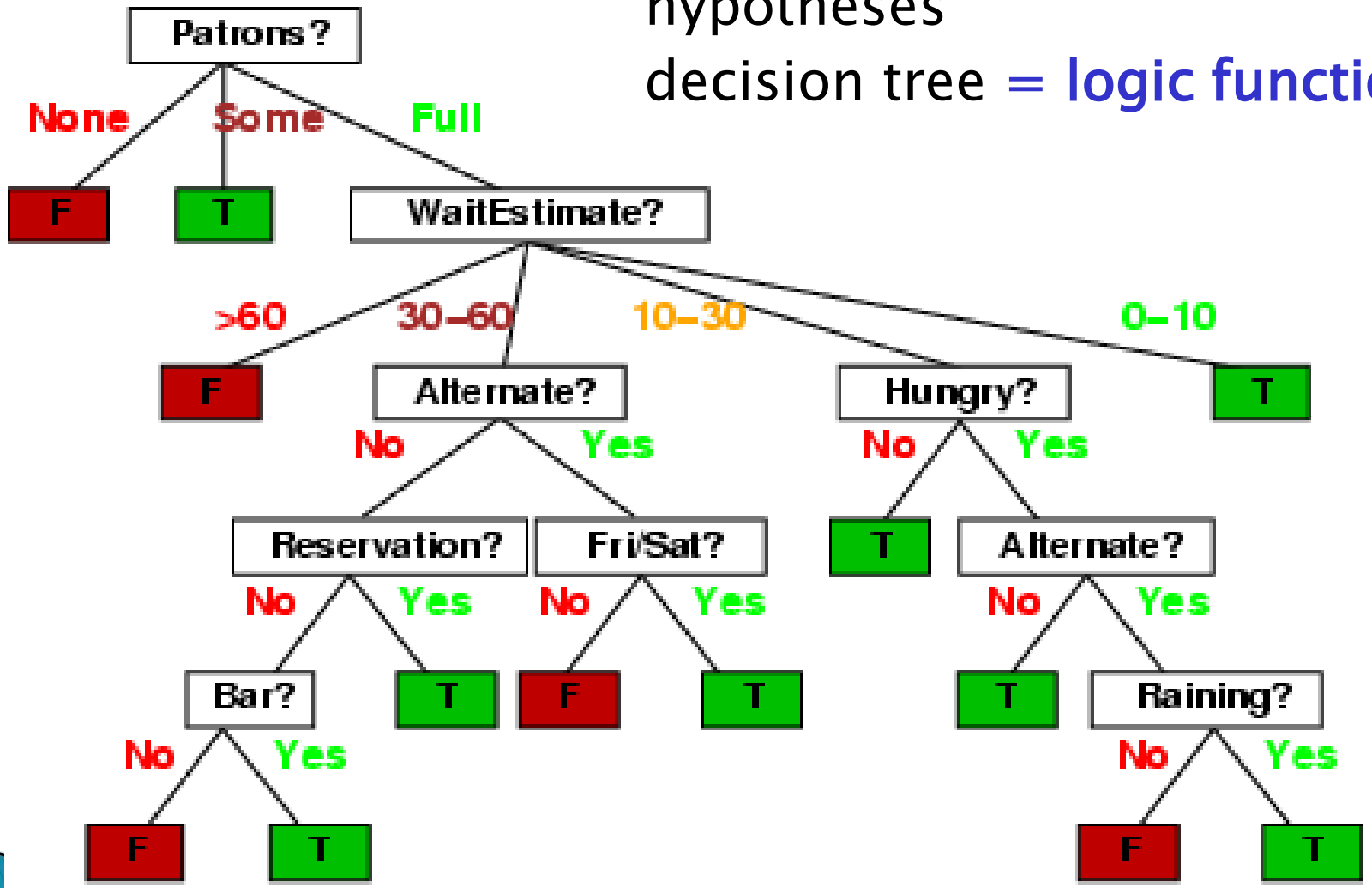
- Examples described by attribute values (Boolean, discrete, continuous)
- E.g., situations where I will/won't wait for a table:
- Classification of examples is positive (T) or negative (F)

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Decision trees

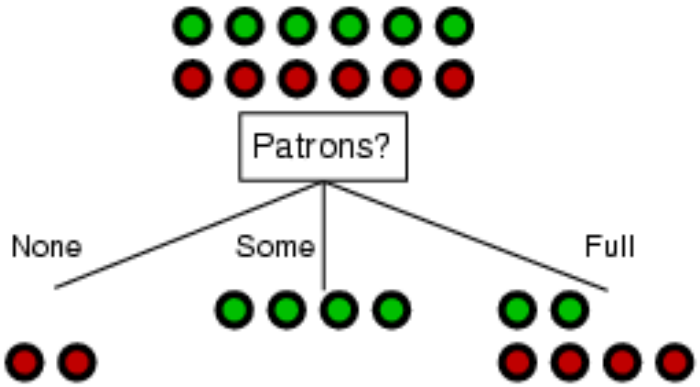
One possible representation for hypotheses

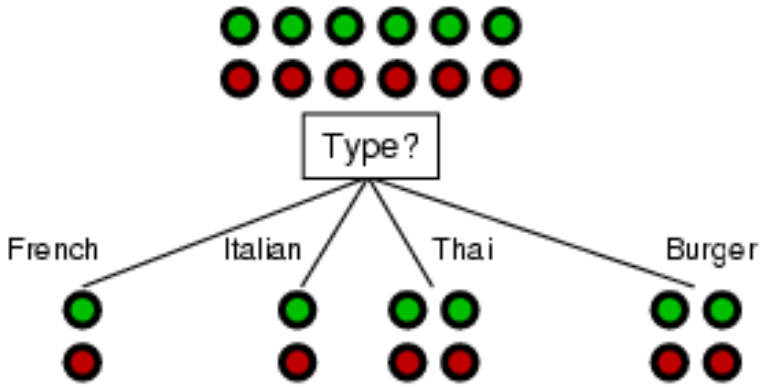
decision tree = logic function



Choosing an attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"





- *Patrons?* is a better choice

Using information theory

- Information Content (Entropy): $I(P(v_1), \ldots, P(v_n)) = \Sigma_{i=1} -P(v_i) \log_2 P(v_i)$
- For a training set containing *p* positive examples and *n* negative examples:

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

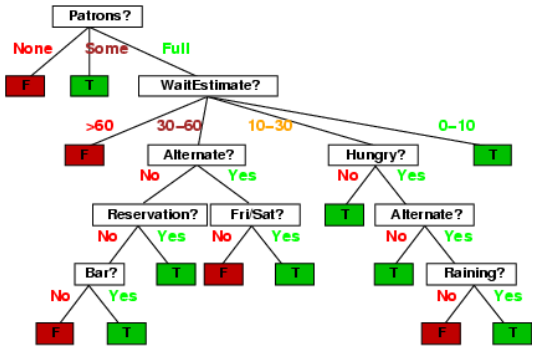
- A chosen attribute *A* divides the training set *E* into subsets $E_1, \ldots, E_v$ according to their values for *A*, where *A* has v distinct values.
- Information Gain (IG) or reduction in entropy from the attribute test:
- Choose the attribute with the largest IG

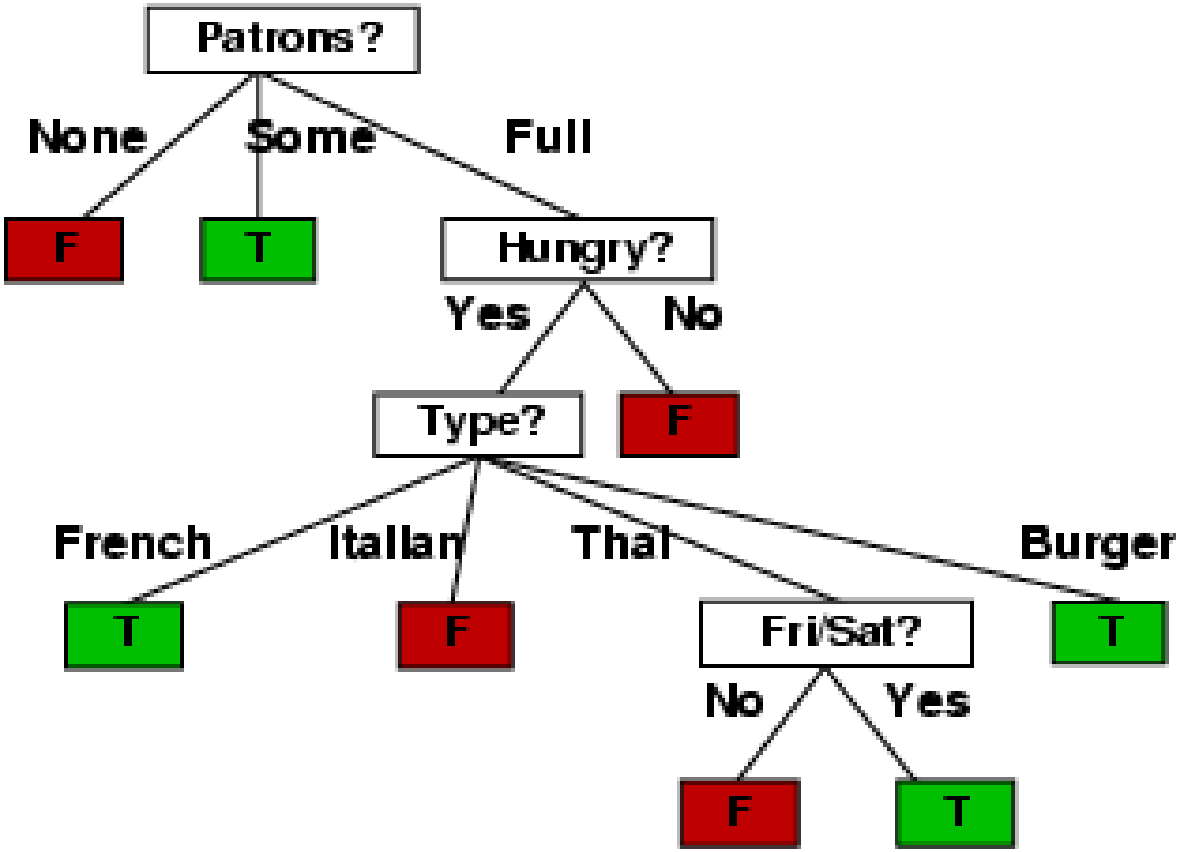
$$remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$$

$$IG(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - remainder(A)$$

Example cont.

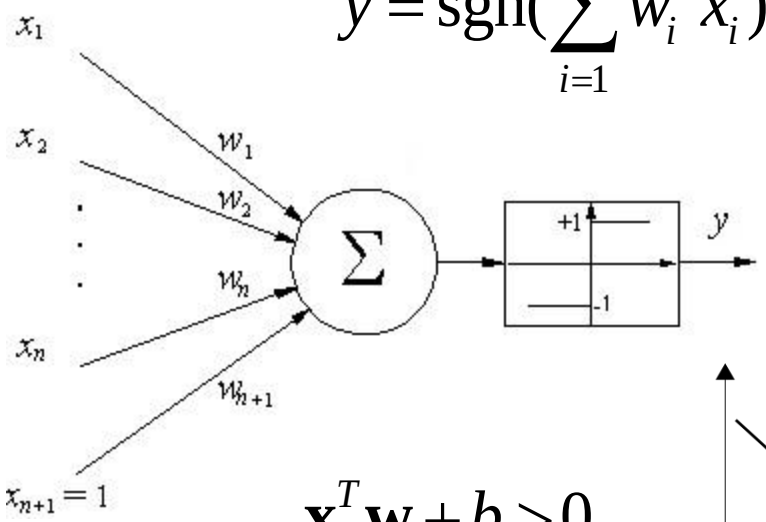
- Decision tree learned from the 12 examples:
- Substantially simpler than “trivial” tree – a more complex hypothesis isn’t justified by small amount of data





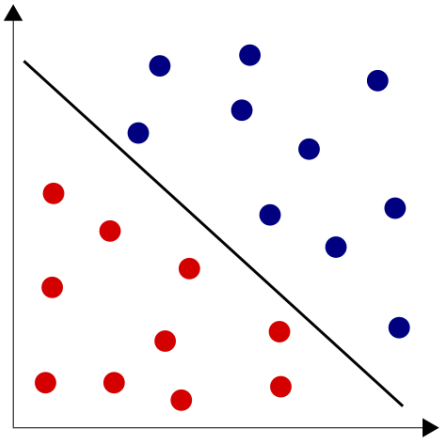
Perceptron

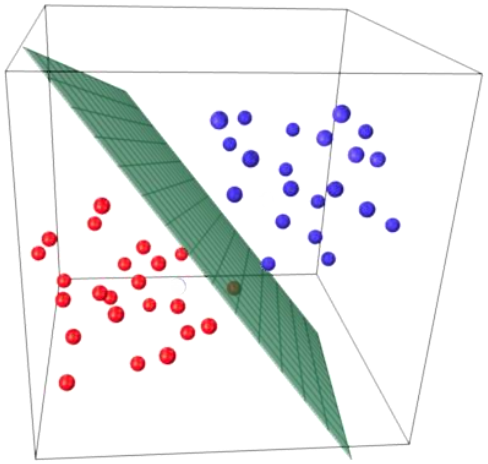
$$y = \mathrm{sgn}\left(\sum_{i=1}^{n+1} w_i\ x_i\right) = \mathrm{sgn}\left(\sum_{i=1}^{n} w_i\ x_i + w_{n+1}\right) = \mathrm{sgn}(\mathbf{x}^T\mathbf{w} + b)$$



$$\mathbf{x}^T\mathbf{w} + b > 0$$

$$\mathbf{x}^T\mathbf{w} + b < 0$$



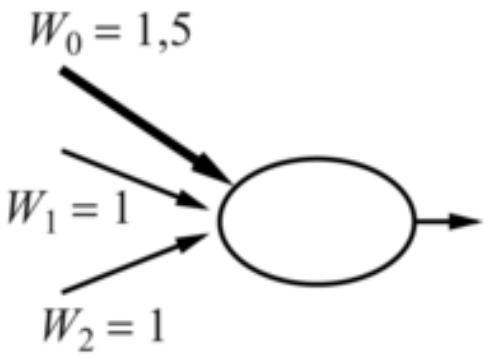


Linearly separable problems

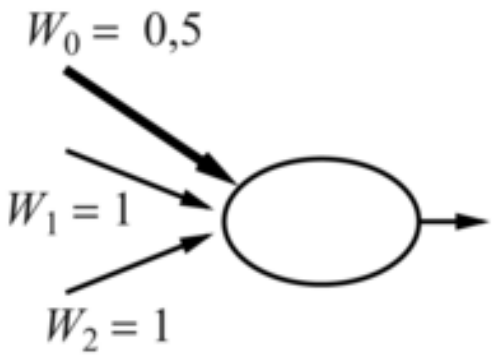
$$\mathbf{w}(k) = \mathbf{w}(k-1) + \alpha\,\varepsilon(k)\,\mathbf{x}(k)$$

α - learning factor

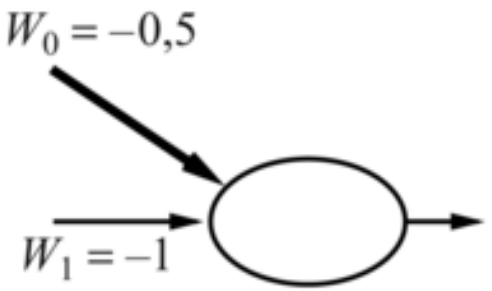
Perceptron and logical functions



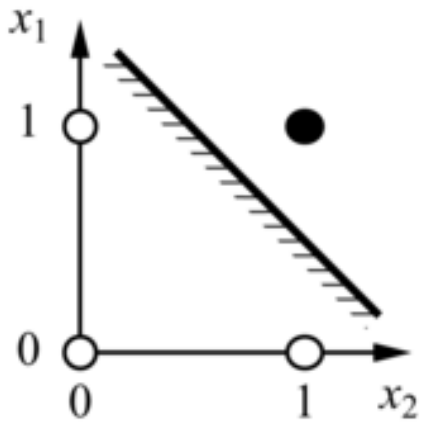
ÉS



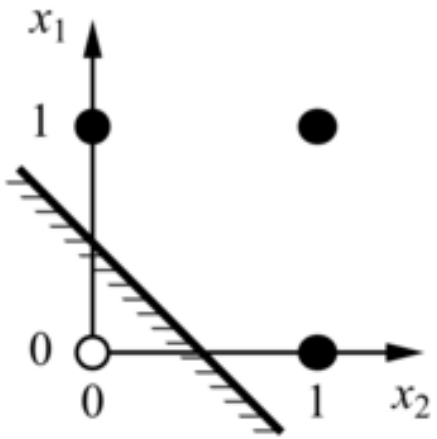
VAGY



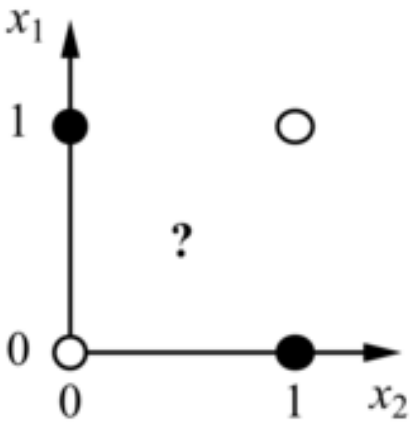
NEM



x_1 és x_2

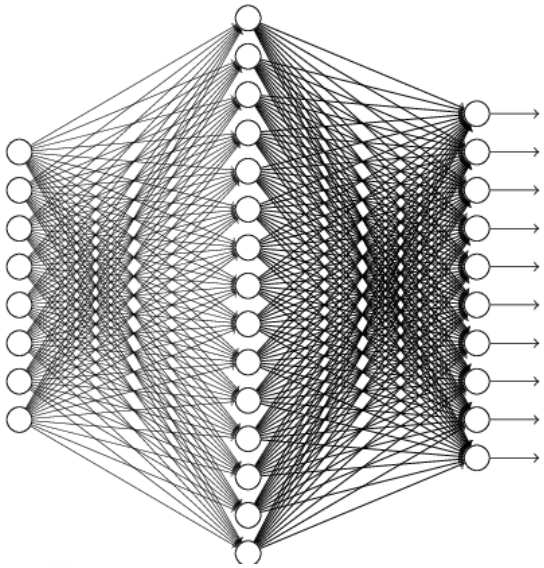


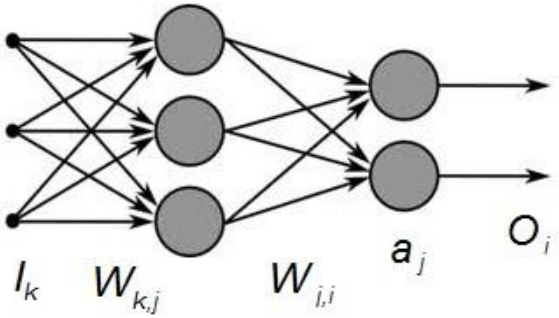
x_1 vagy x_2

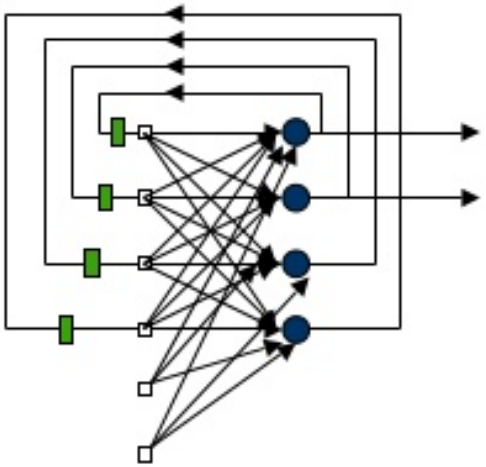


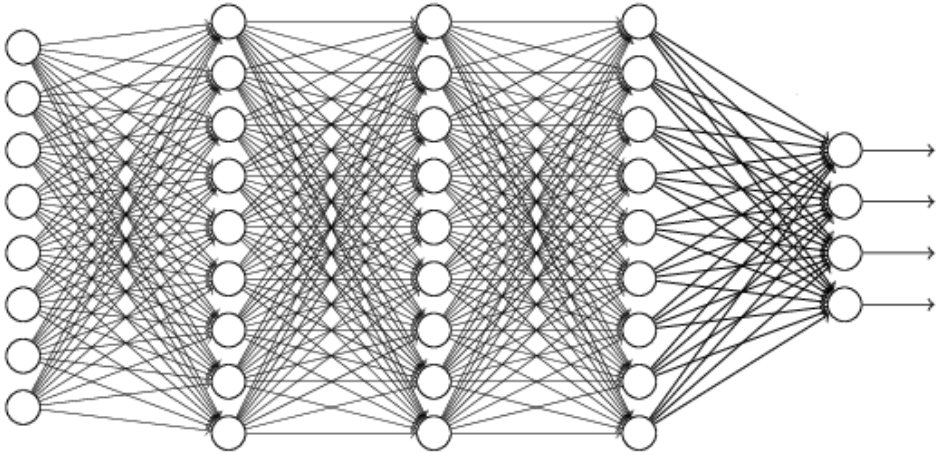
x_1 xor x_2

Artificial Neural Network

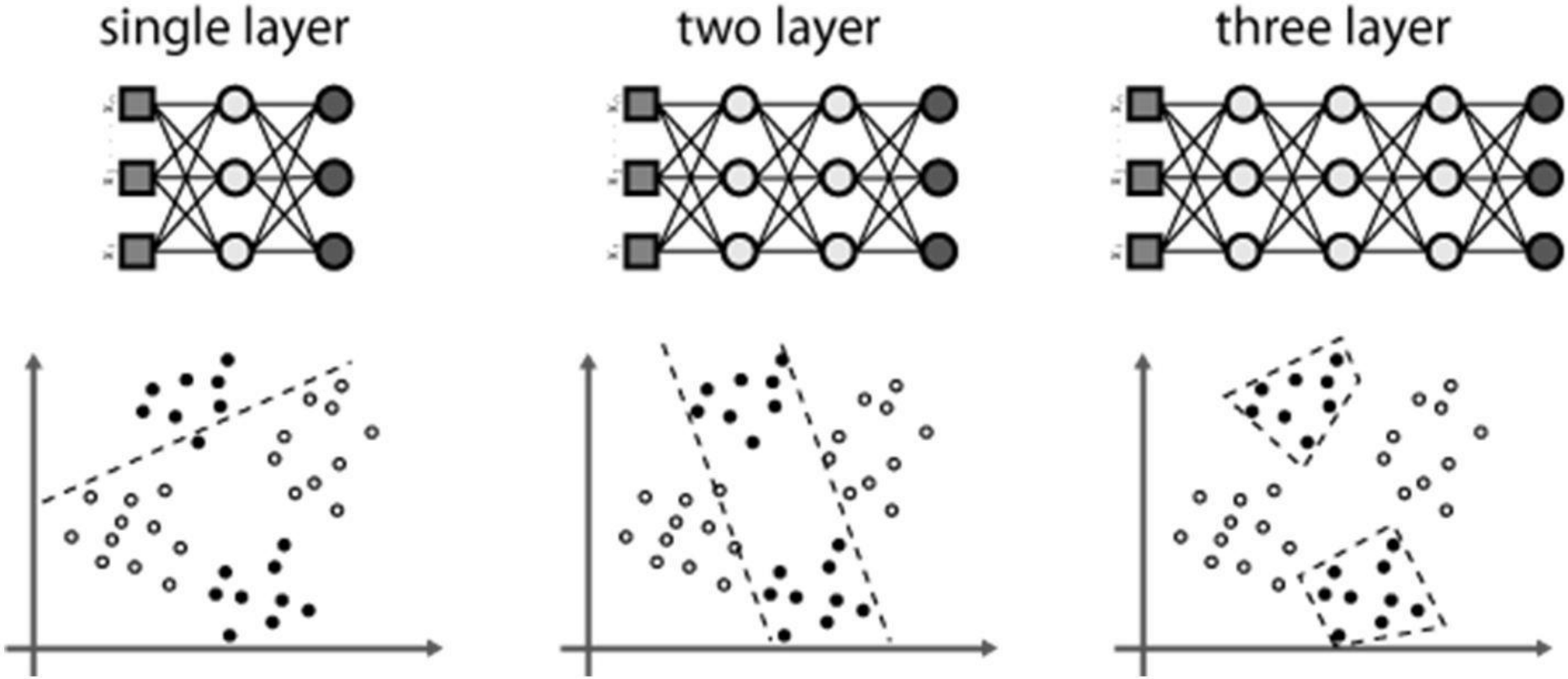






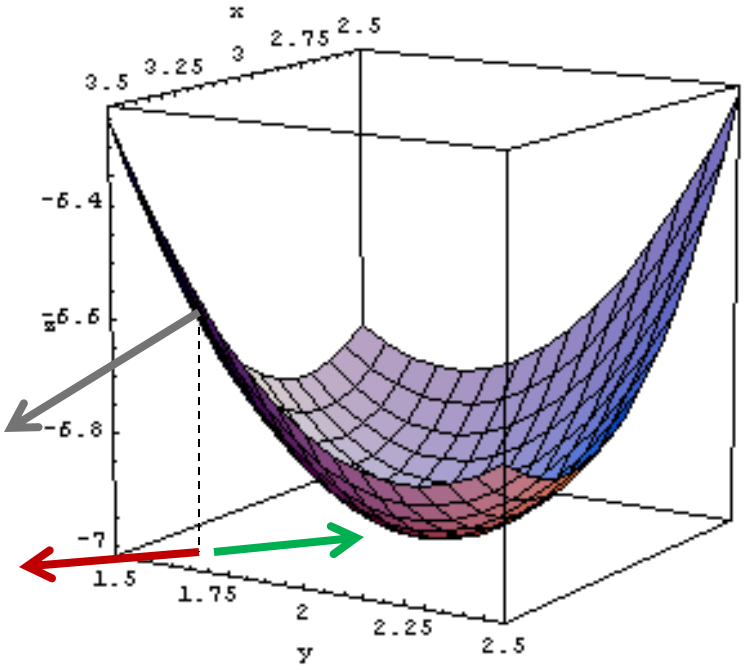


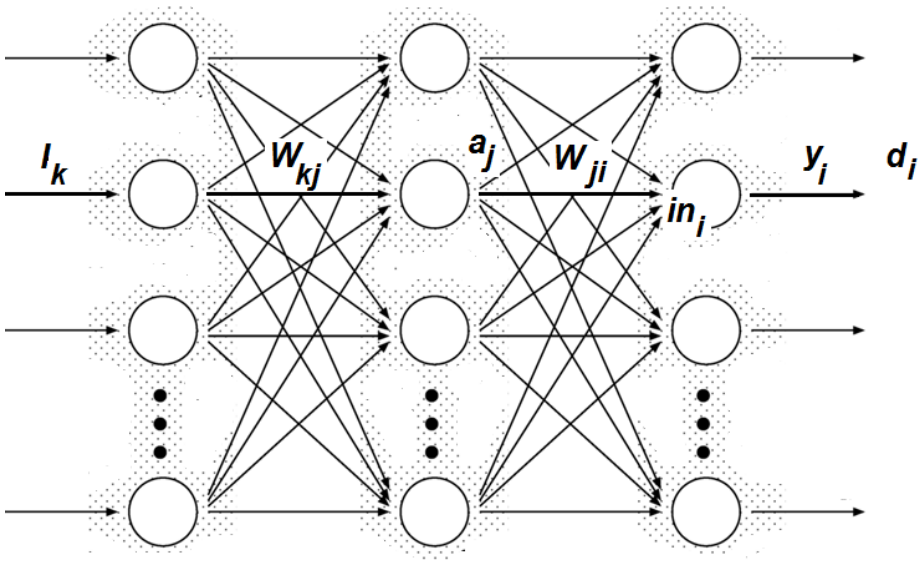
Artificial Neural Network



Artificial Neural Network

$$W \leftarrow W - \alpha \frac{\partial E}{\partial W}$$





$$E = \frac{1}{2}\sum_i (d_i - y_i)^2$$

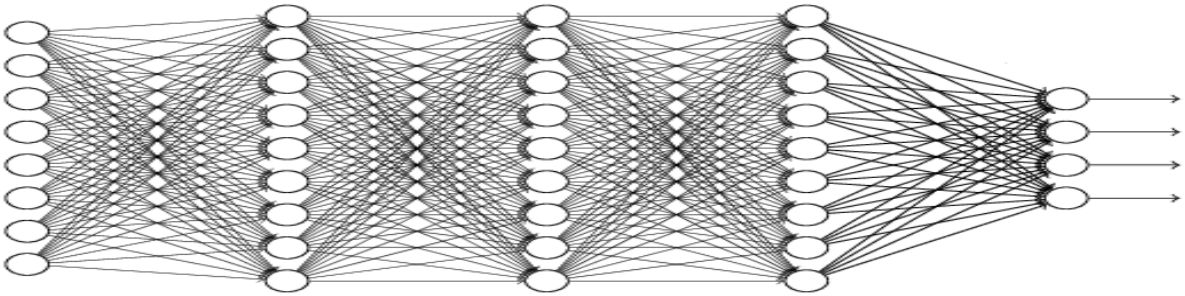
$$W_{j,i} \leftarrow W_{j,i} - \alpha \frac{\partial E}{\partial W_{j,i}}$$

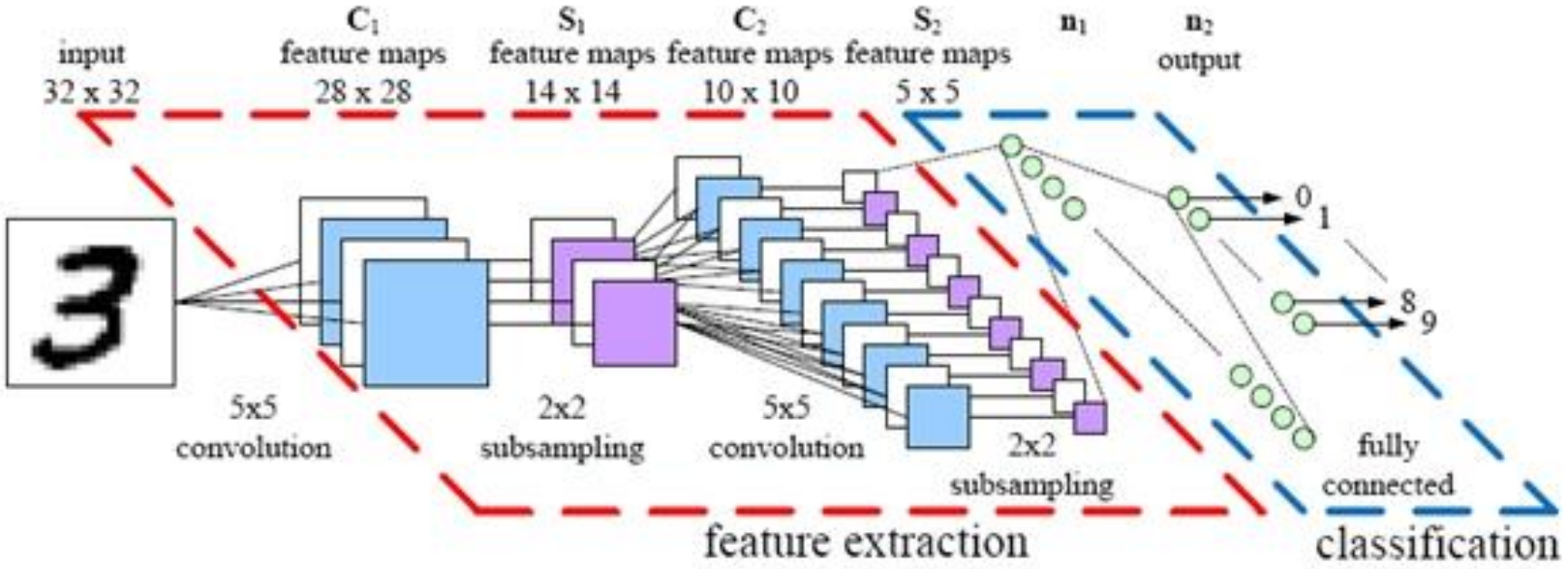
Error backpropagation algorithm

$$W_{k,j} \leftarrow W_{k,j} - \alpha \frac{\partial E}{\partial W_{k,j}}$$

Artificial Neural Network

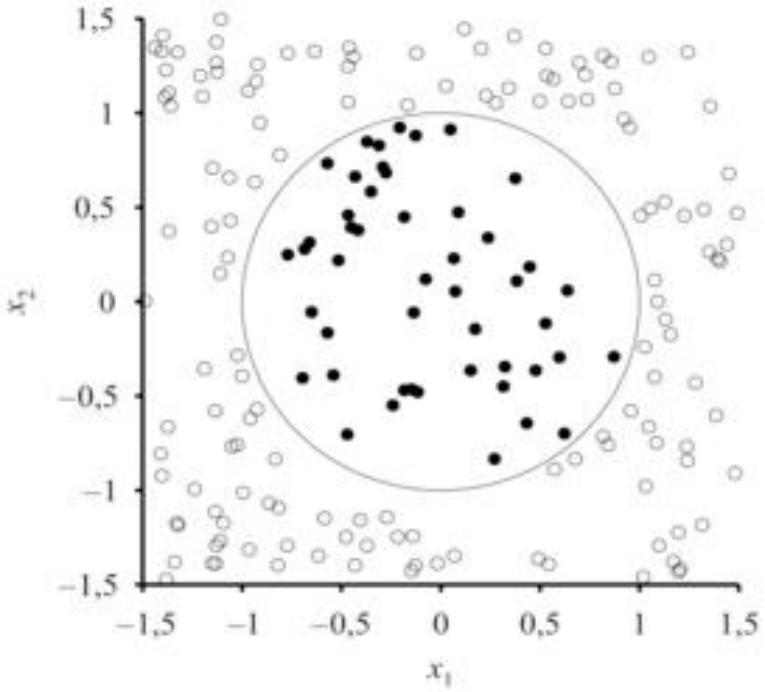
Deep Learning with Convolutional Networks (CNN)



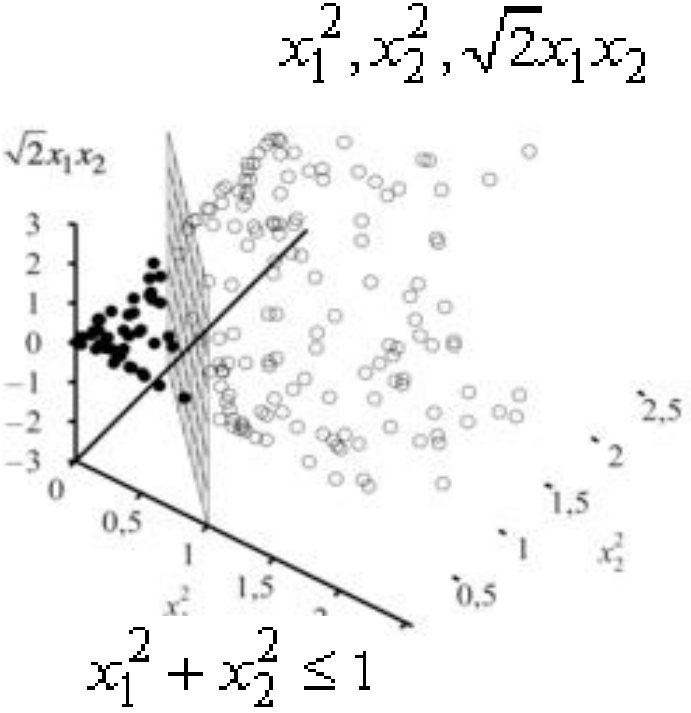


Kernel algorithms (SVM, support vector machines)

Kernel trick



(a)



(b)

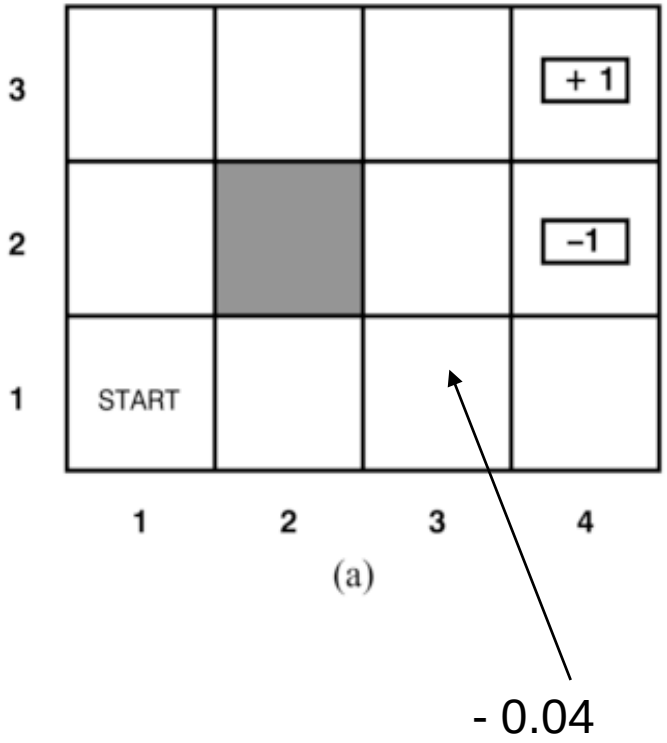
$$\varphi : \mathbf{x} = (x_1, x_2) \rightarrow \varphi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

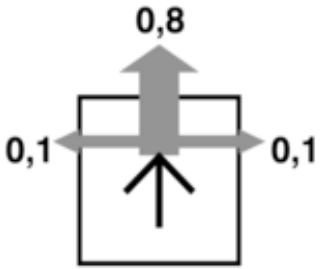
$$\varphi(\mathbf{z})^T \cdot \varphi(\mathbf{x}) = (z_1^2, z_2^2, \sqrt{2}z_1z_2) \cdot (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

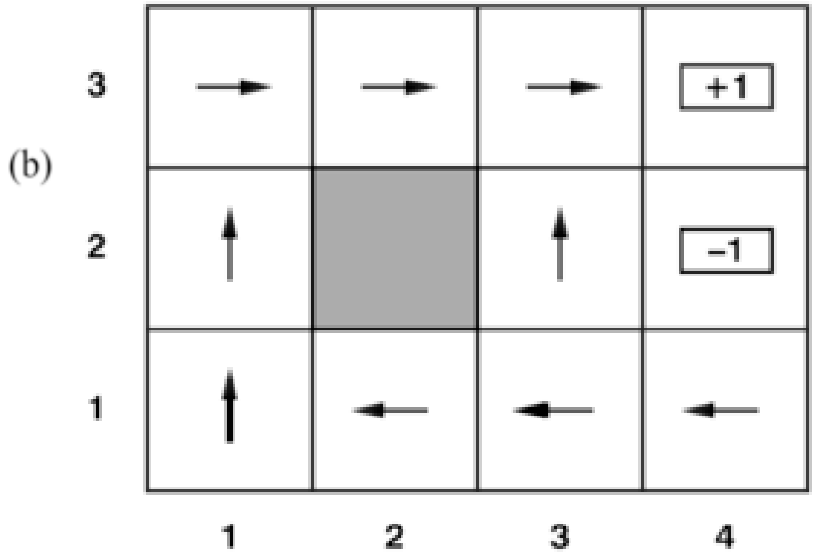
$$= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 = (x_1z_1 + x_2z_2)^2$$

$$= (\mathbf{z}^T \cdot \mathbf{x})^2 = k(\mathbf{z}, \mathbf{x})$$

Sequential decision problem







Up, up, right, right, right?

(optimal) Policy: $\pi(s) = a$

Sequential decision problem

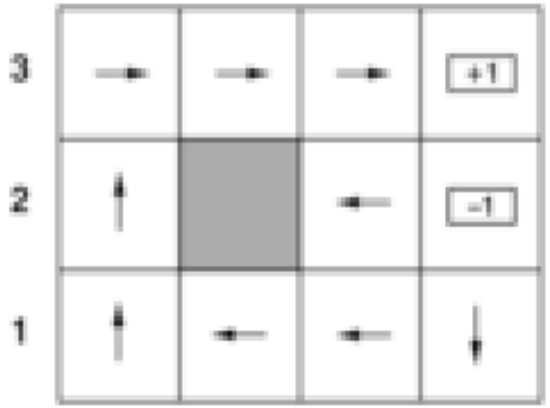
Markov decision process

Starting state: S0

State-transition model: T(s, a, s′)

Reward: R(s), vagy R(s, a, s′)

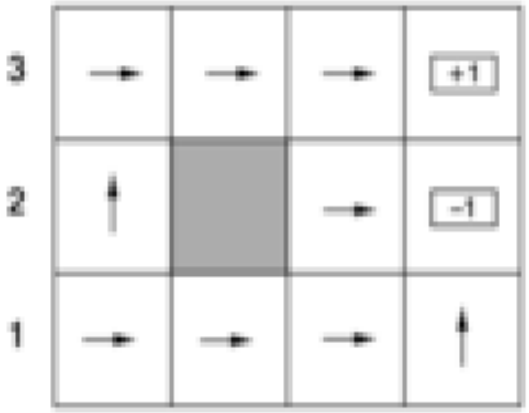
– 0,0221 < R(s) < 0



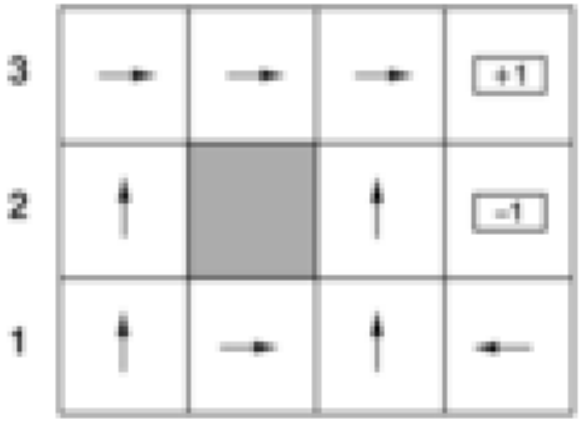
Optimal policy =
folyamatosan choosing
optimal movement,
decision, action $\pi(s) = a$

$\pi^*(s) = a$

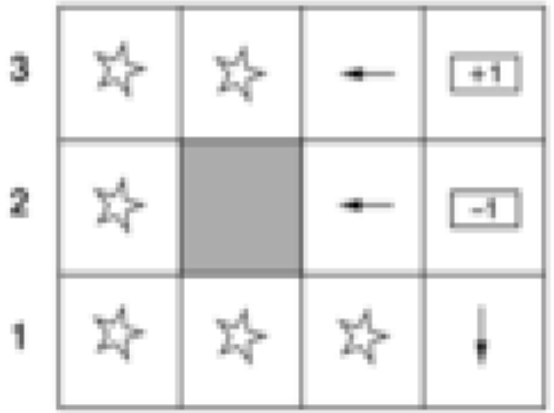
R(s) ≤ –1,6284



– 0,4278 ≤ R(s) ≤ – 0,0850



R(s) > 0



Sequential decision problem

Reward discount $U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$

Utility of state (**Bellman**) $U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s') U(s')$

Bellman-updating $U_{t+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s') U_t(s')$

Policy iteration

$U_t(s) = R(s) + \gamma \sum_{s'} T(s, \pi_t(s), s') U_t(s')$

$U_{t+1}(s) \leftarrow R(s) + \gamma \sum_{s'} T(s, \pi_t(s), s') U_t(s')$

for each s állapotra **in** S **do**

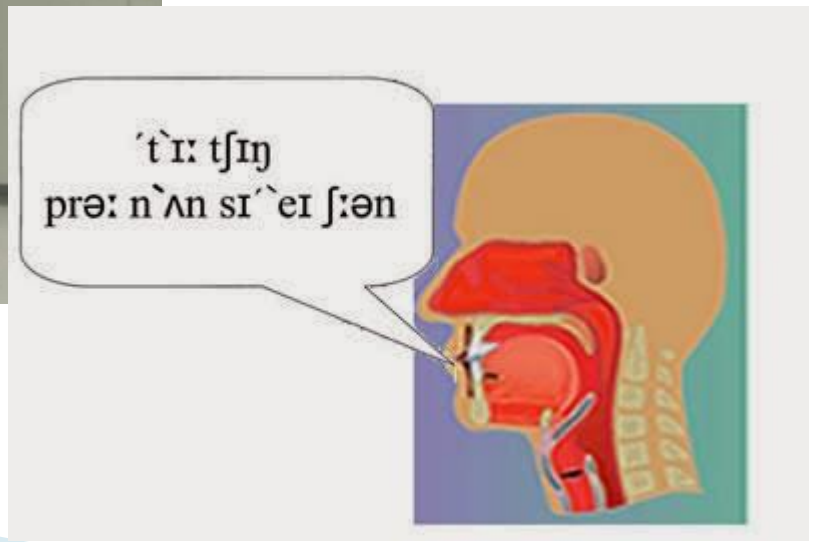
if $\max_a \sum_{s'} T(s,a,s') U[s'] > \sum_{s'} T(s, \pi[s], s') U[s']$ **then**

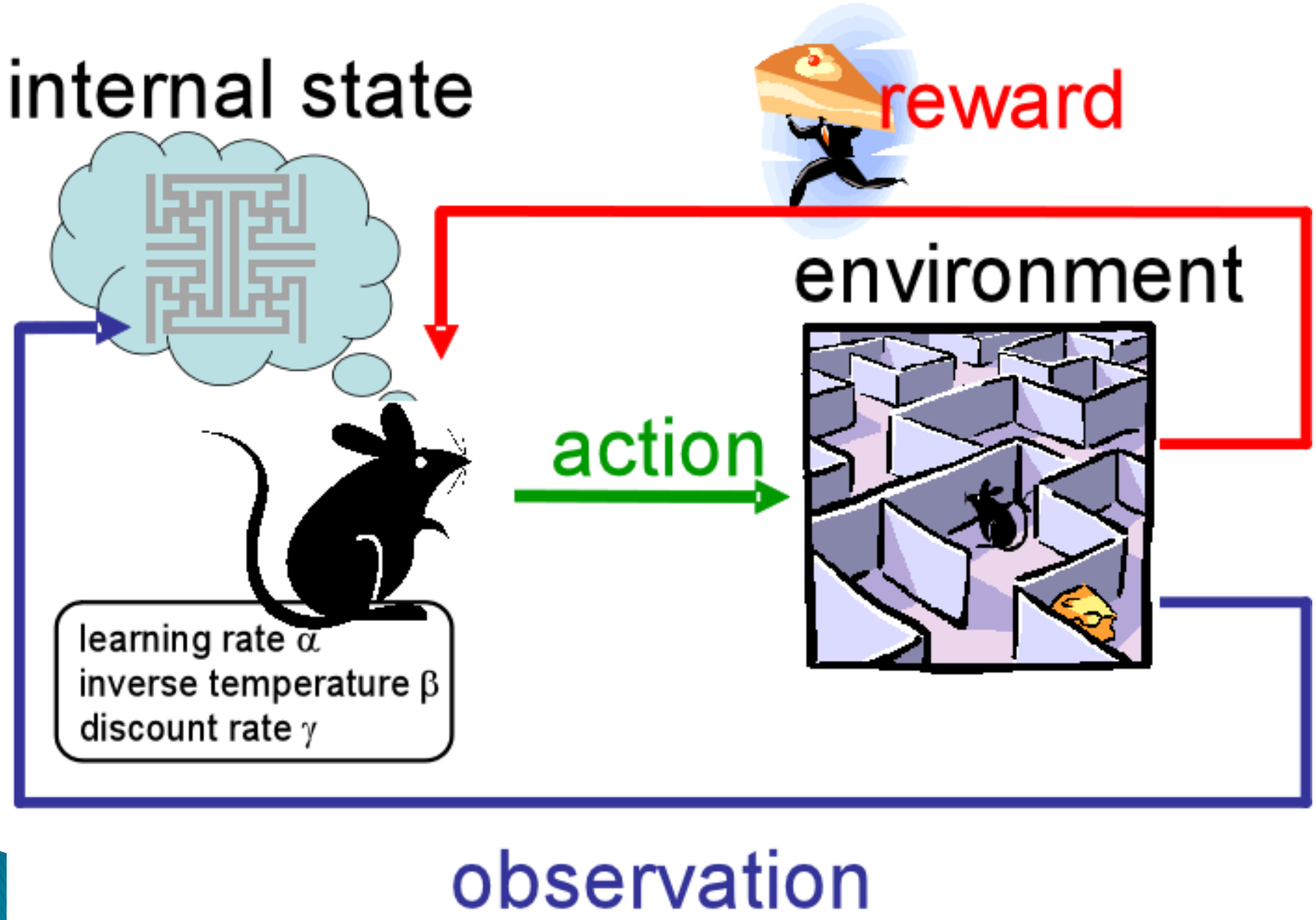
$\pi[s] \leftarrow \text{argmax}_a \sum_{s'} T(s,a,s') U[s']$

Starting from sequential decision problem

Reinforcement learning

Init state:	known
State-transition model:	**not known, experience at most**
Rewardfunction:	**not known, at most as experienced**
Optimal policy	**to be learned, again the odds?!**




internal state
reward
environment
action
learning rate α
inverse temperature β
discount rate γ
observation

Agent knowledge:

Environment known upon start, or it must be also learned.

Reinforcement:

Only in the goal state, or also in other states under way.

Agent:

Pasive learner: observers the environment and learns.

Active learner: must act upon learned information.

exploring …

Learning U(s) **utility** function, deciding actions on this basis, that the expected utility gain be maximized (environment/agent model **needed**)

Learning Q(a, s) **action value** function (state-action pairs), relating some utility to an action in a given situation (environment/agent model **not needed**, learned meantime) – **Q learning (model-free)**

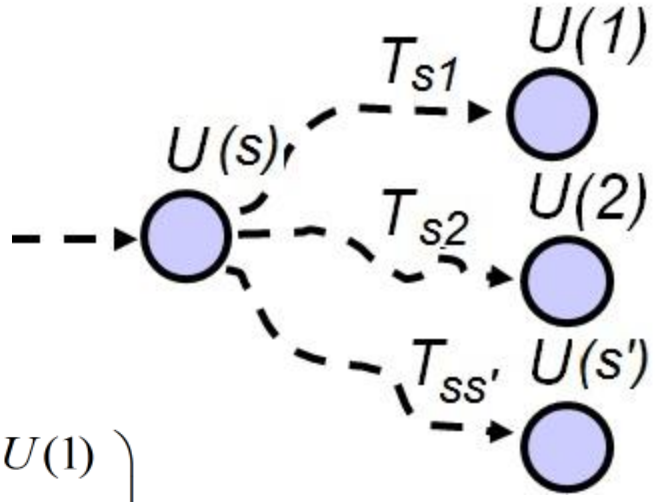
Expected utility of a state = a reward-to-go – discounted additive utility of the coming states

$$U^{\pi}(s) = E[\sum_t \gamma^t R(s_t) \,|\, \pi, s_0 = s]$$

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U^{\pi}(s')$$

observed

observed



$$\begin{pmatrix} U(1) \\ U(2) \\ U(3) \\ \dots \\ U(N) \end{pmatrix} = \begin{pmatrix} R(1) \\ R(2) \\ R(3) \\ \dots \\ R(N) \end{pmatrix} + \gamma \begin{pmatrix} T_{11} & T_{12} & T_{13} & \dots & T_{1N} \\ T_{21} & T_{22} & T_{23} & \dots & T_{2N} \\ T_{31} & T_{32} & T_{33} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ T_{N1} & T_{N2} & \dots & \dots & T_{NN} \end{pmatrix} \begin{pmatrix} U(1) \\ U(2) \\ U(3) \\ \dots \\ U(N) \end{pmatrix}$$

$$\mathbf{U} = \mathbf{R} + \gamma\, \mathbf{T}\mathbf{U} \qquad \mathbf{U} = (\mathbf{I} - \gamma \mathbf{T})^{-1} \mathbf{R}$$

Adaptive Dynamic Programing

TD – Time Difference learning

Idea: let us use the estimated utilities together with observed transitions:

TD(0)-difference (utility estimate)

$$R(s)+\gamma U(s')$$

TD(0) -error:

$$\delta(s)=R(s)+\gamma U(s')-U(s)$$

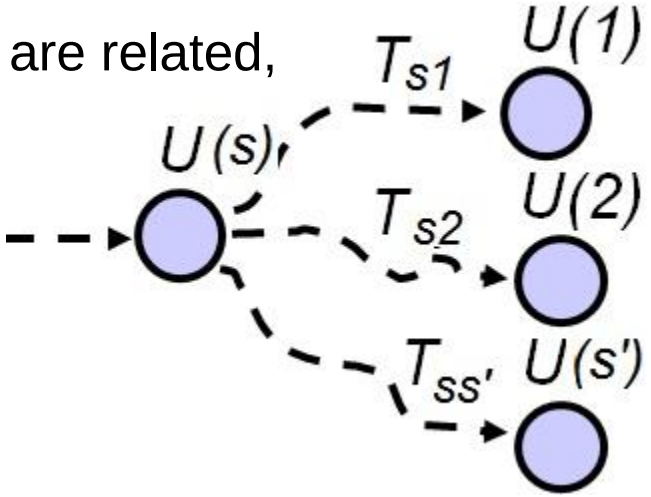
$$U(s)\leftarrow(1-\alpha)U(s)+\alpha\,\delta(s)=U(s)+\alpha\,(R(s)+\gamma U(s')-U(s))$$

α- **learning factor,** γ – **discount factor**

Difference in utility of the following-up states: **temporal difference**, **TD**.

The TD is using the information how the states are related, but only that which is coming from **the actually observed sequence**.

TD can be used in **an unknown environment** also.



Active learning in unknown environment

Decision: which action? What are the effects of this action?
How they do influence the reward?

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$$

What about the **influence of the action on the learning process?**.
Decision has two effects:

1. Yields reward in the **actual** sequence.
2. Affects the observations and thus the **agent learning capability**
   – so it affects rewards in **future sequences**.

Exploring the state space

Trade-off: actual reward and **long term** advantages.

„**Explorer**„: acts randomly exploring the whole environment. (**exploration**)
„**Greedy**„: maximizes gains based on actual utility estimate. (**exploitation**)

Let the agent be explorer when the knowledge about the environment is minute, and let it be greedy, when it has already a good model.

Exploring functions

ε-greedy: agent acts randomly with probability ε, and greedy action with probability 1-ε

Boltzmann-exploring model
The probability of chosing an action in s state:

T „temperature” controls both extremes.
If $T \rightarrow \infty$, then the choice is purely (uniformly) random,
if $T \rightarrow 0$, then the choice is greedy.

$$P(a,s) = \frac{e^{utility(a,s)/T}}{\sum_{a'} e^{utility(a',s)/T}}$$

Q-learning

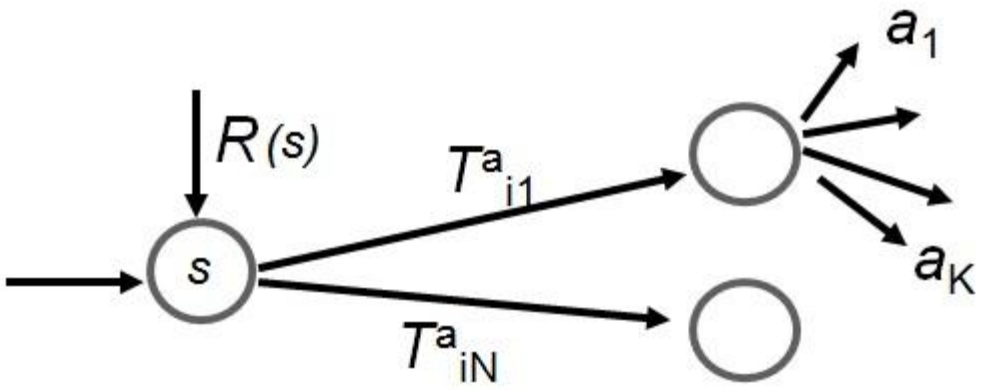
Utility of an action chosen in a given state: **Q-value**

$$U(s) = \max_a Q(a, s)$$

Importance of Q-value:
decision possible **without using the model**,
can be learned **directly** from the reward feedback.

Equilibrium equation, valid for the correct Q-values:

$$Q(a, s) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(a', s')$$



Q-learning

Direct usage: iteration computing true Q values (model!)

Time Difference method requires no model:

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

SARSA Q-learning (State-Action-Reward-State-Action)

$$Q(a,s) \leftarrow Q(a,s) + \alpha[R(s) + \gamma\, Q(a',s') - Q(a,s)]$$

(a' chosen e.g. from the Boltzmann exploring model)

Generalization capability of reinforcement learning

The learned utility: storage as a **table** = **explicit reprezentáció**

Well designed approximated ADP: 10000, or more.
Real-life state spaces – vastly larger. (chess state space ca. 10^{50}-10^{120})

The only possibility is the **implicit reprezentation** of the function:

E.g. in a game a set of the properties of the game state (board state) $f_1,\ldots f_n$.
The estimated utility of the board (state):

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots \theta_n f_n(s)$$

The utility function can be chracterized by *n* values, instead of e.g. 10^{120} values.
An average chess utility (evaluation) function has app. 10 weights, *enormous* compression and reduction.

Implicit reprezentation makes it possible that due to the reduction the learning agent can generalize from the visited state to the as yet not visited states.

The most important aspect of the implicit reprezentation is the so called **inductive generalization of the input states**.

Using **TD** for inductive implicit learning.
U or *Q* table → implicit reprezentation (e.g. neural net).

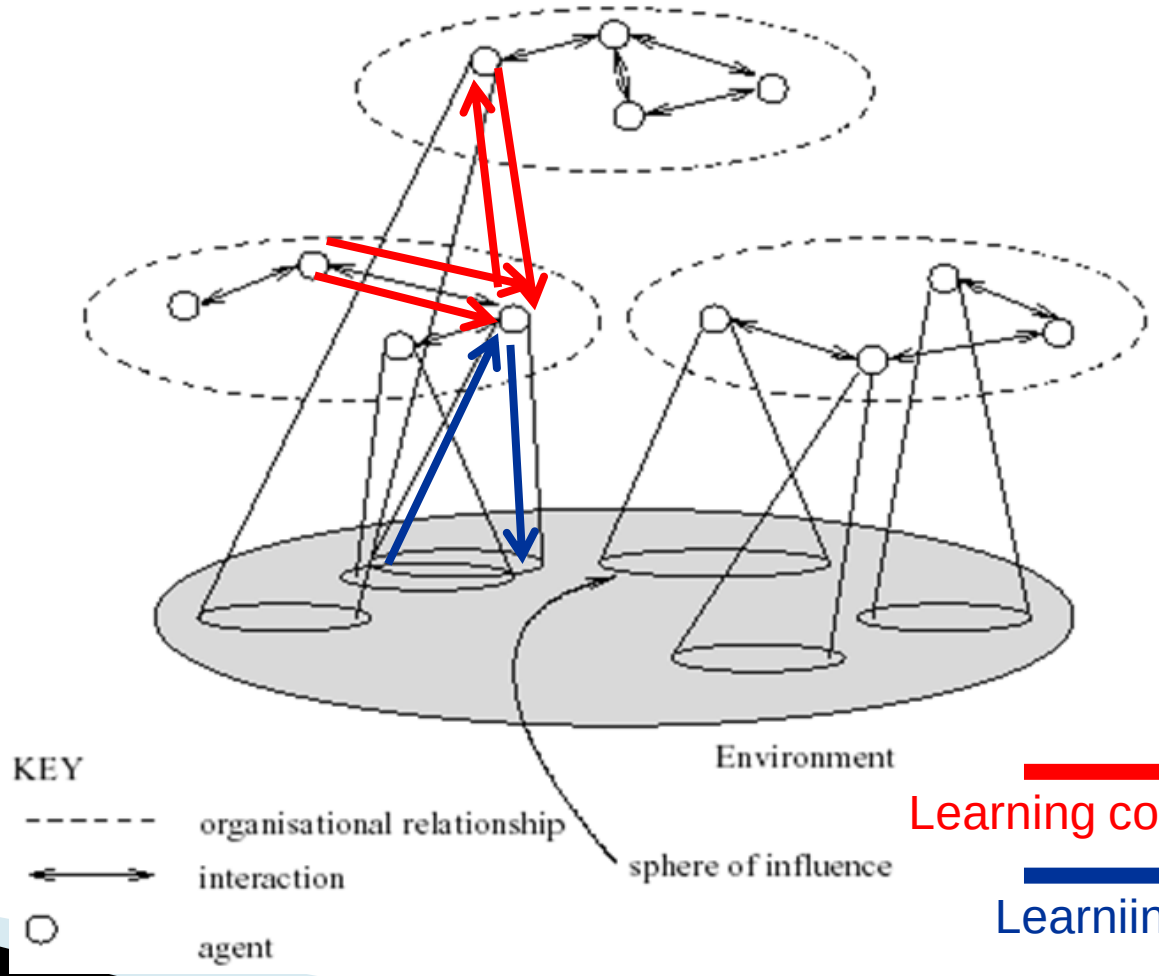
U/*Q* learning gradients

$$\theta_i \leftarrow \theta_i + \alpha \left[R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s) \right] \frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

$$\theta_i \leftarrow \theta_i + \alpha \left[R(s) + \gamma \max_{a'} \hat{Q}_\theta(a', s') - \hat{Q}_\theta(a, s) \right] \frac{\partial \hat{Q}_\theta(a, s)}{\partial \theta_i}$$

MAS learning - facts/challenges

Hypothesis space
Emergency
Game theory perspective
Critics
Reinforcement
Stationarity
Dynamics
Merit assignment
Nash-equilibrium
Convergence
Communication



MAS learning - facts/challenges

Learning from each other
Learning about each other
Learning despite others
Learning with the help of others

Learning from organization

Learning when interacting with changing entities

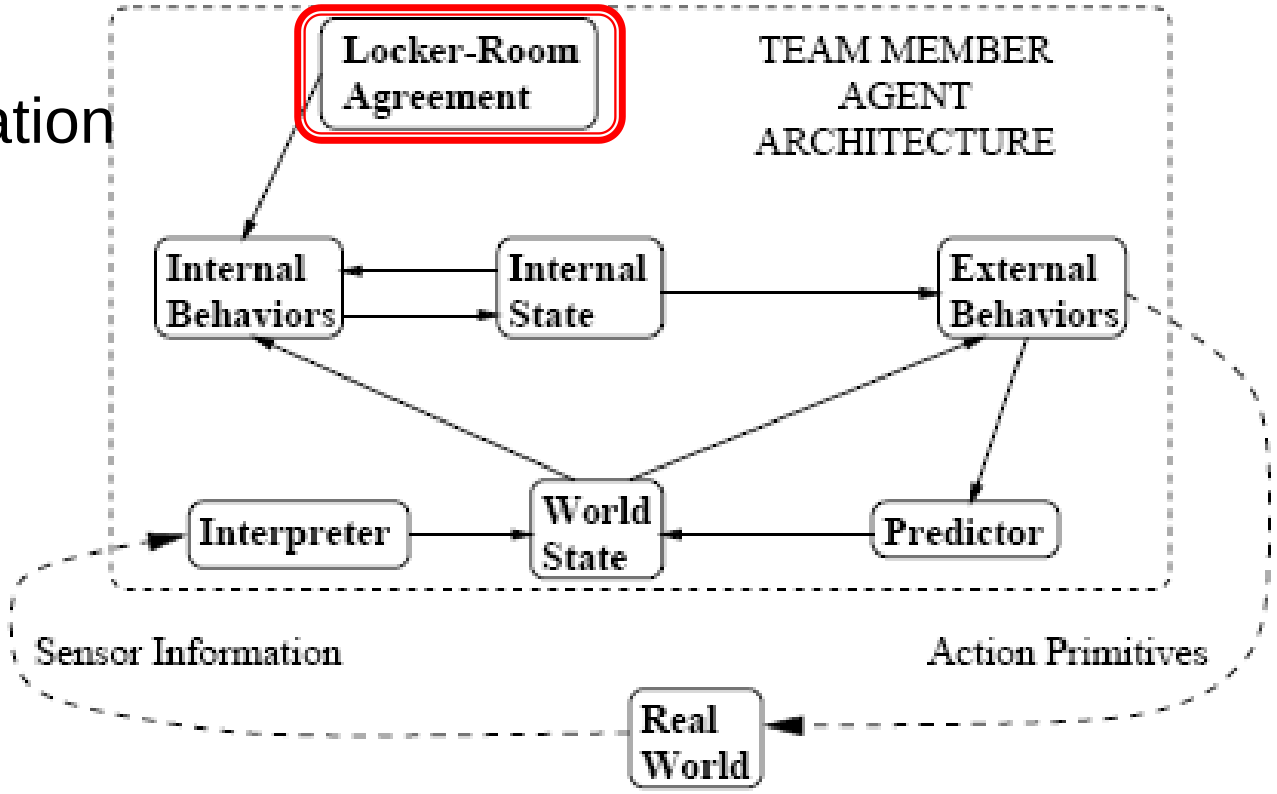
Forgetting group partners

Learning form negotiation
Learning during negotiation
Learning after concluding negotiation

An organization in an adverse natural environment, or against other organizations

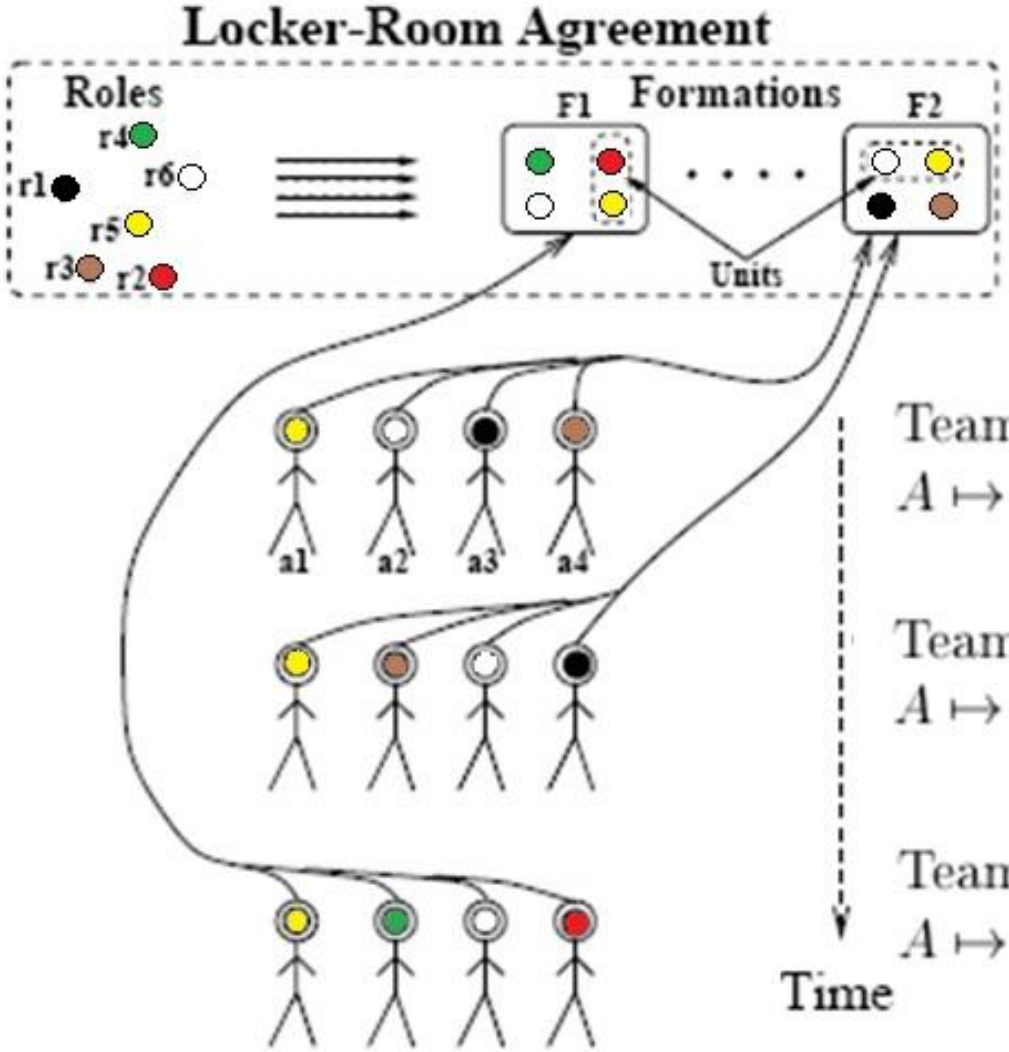
Dynamic, „adverse” (non-cooperative), real-time environment, with **limited communication**

PTS – Periodic Team Synchronization



Formations

$$F = \{R, \{U_1, U_2, ..., U_k\}\}$$



F1 = {r2,r4,r5,r6,{r4,r5}}
F2 = {r1,r3,r5,r6,{r5,r6}}

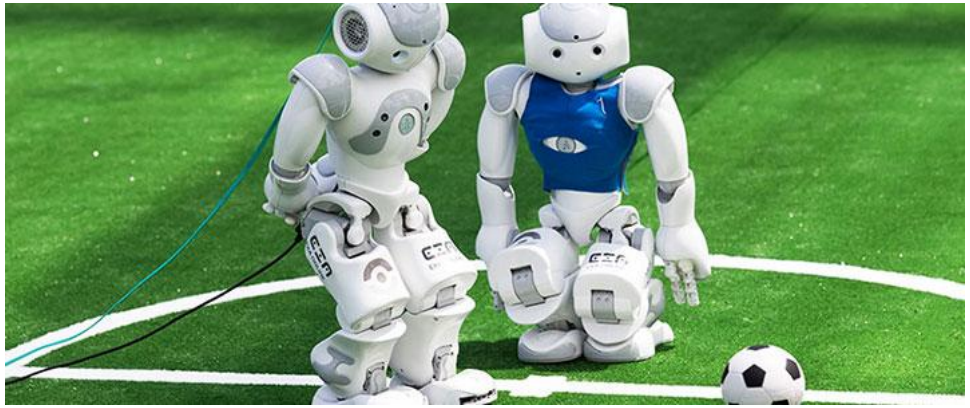
Team Formation = F2
$A \mapsto R = \{(a1,r5),(a2,r6),(a3,r1),(a4,r3)\}$

Team Formation = F2
$A \mapsto R = \{(a1,r5),(a2,r3),(a3,r6),(a4,r1)\}$

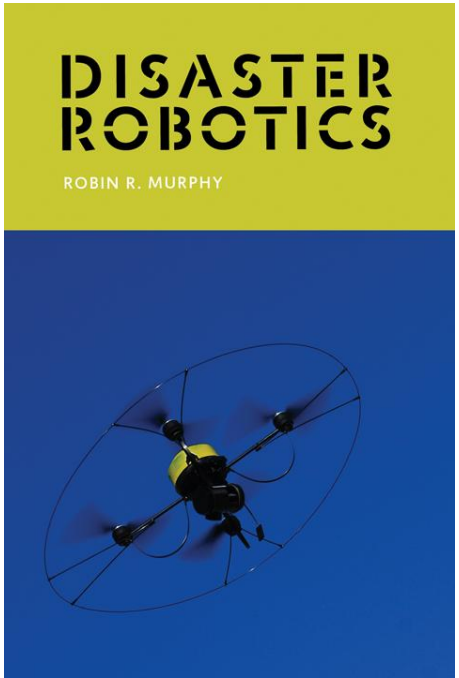
Team Formation = F1
$A \mapsto R = \{(a1,r5),(a2,r4),(a3,r6),(a4,r2)\}$

Robot World Cup

RoboCup-97 Nagoya



RoboCup-Rescue

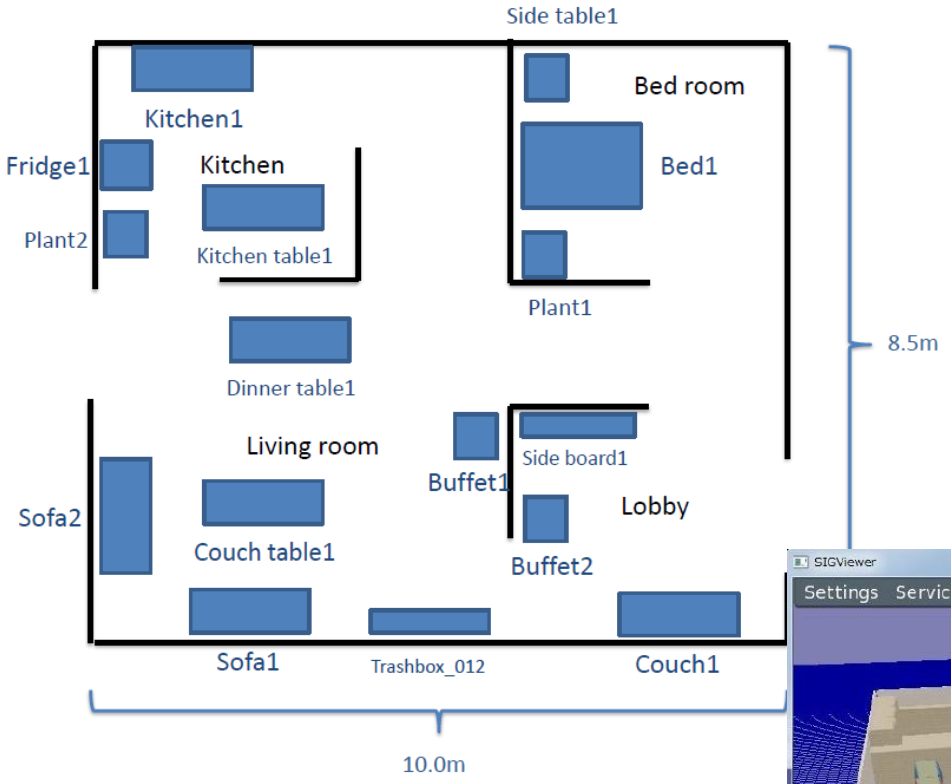


USAR (Urban Search And Rescue) Arenas_ Test Arenas for Autonomous Mobile Robots_

http://www.nist.gov/el/isd/testarenas.cfm

Specific robotic capabilities:

- Negotiate compromised and collapsed structures
- Locate victims and ascertain their conditions
- Establish communications with victims
- Deliver fluids, nourishment, medicines
- Emplace sensors to identify/monitor hazards
- Mark or identify best paths to victims



RoboCup@Home arena

Follow me, Clean up, …

„Move to the LOCATION, find a person, and guide it to the exit. …" stb.

EGPSR

Endurance General Purpose Service Robot Test

Multi Agent Reinforcement Learning

.

$$\left(N, S, A = \underset{k \in N}{\times} A_k, T, \{R_k\}\right), \quad T : S \times A \times S \to [0,1],\ R_k : S \times A \to \Re$$

$$R_i(s, \mathbf{a})$$

Reinforcement:

- **fully cooperative** agent system $\rho_1 = \ldots = \rho_n$
- **fully competetive** agent system $\rho_1 = -\rho_2$ $\rho_1 + \rho_2 + \ldots \rho_n = 0$
- **mixed** $\rho_1 + \rho_2 + \ldots + \rho_n > 0$

Aim of learning?

(1) Stability – convergence to some equilibrium (e.g. NE)
(2) **Adaptivíty** – successfully learn the adversary strategy.
(3) A particular gain in utility.

From one agent to multi agents

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left[r_{k+1} + \gamma \max_{a'} Q_k(s_{k+1}, a') - Q_k(s_k, a_k) \right]$$

Actions of others

$$Q_{k+1}(s_k, \mathbf{a}_k) = Q_k(s_k, \mathbf{a}_k) + \alpha_k \left[r_{k+1} + \gamma \max_{\mathbf{a}'} Q_k(s_{k+1}, \mathbf{a}') - Q_k(s_k, \mathbf{a}_k) \right]$$

Something different to account for the friendly or the adverse character of the others to compute the future.

$$Q_{k+1}(s_k, \mathbf{a}_k) = Q_k(s_k, \mathbf{a}_k) + \alpha_k \left[r_{k+1} + \gamma \left[XYZ \right] - Q_k(s_k, \mathbf{a}_k) \right]$$

1 agent ⇨ 2 agent ⇨ N agent

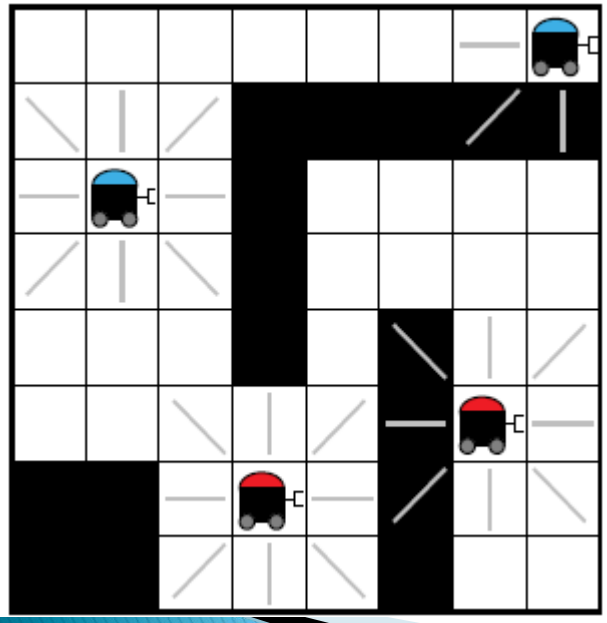
Multi Agent Deep Learning

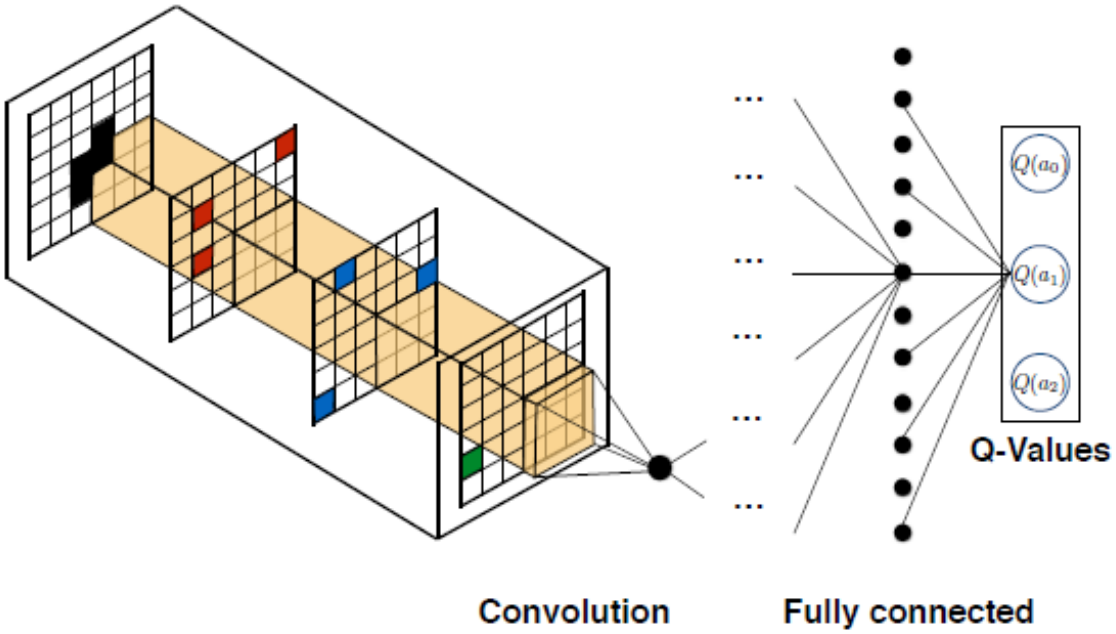
$$Q(s,\mathbf{a}) = Q(s,\mathbf{a}) + \alpha\left[r + \gamma \max_{a'} Q(s',\mathbf{a}') - Q(s,\mathbf{a})\right], \quad Q(s,\mathbf{a}|\boldsymbol{\theta})$$

$$\min L(s,\mathbf{a}|\boldsymbol{\theta}_i) = \left(r + \gamma \max_a Q(s',\mathbf{a}|\boldsymbol{\theta}_i) - Q(s,\mathbf{a}|\boldsymbol{\theta}_i)\right)^2$$

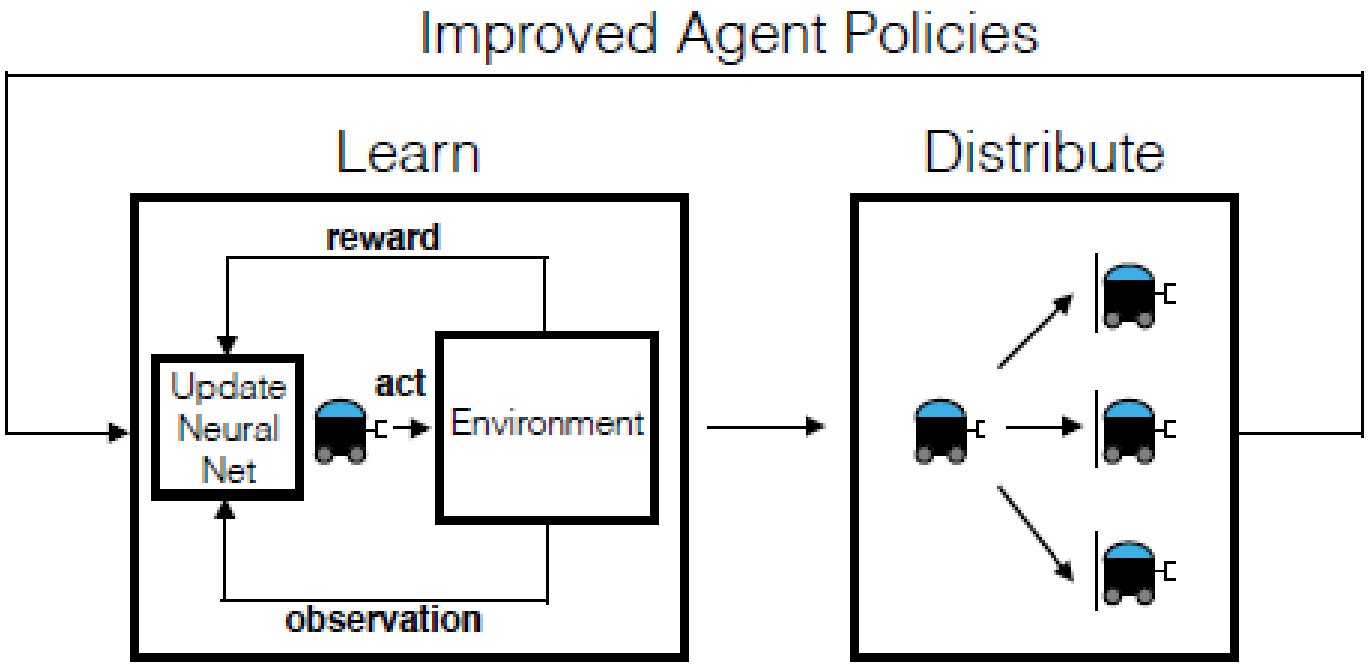
$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_i)$$

Environment 2 dim, 2 agent type.
Convolution NN channels: background (obstackles), enemy, aliens, agent own:
4 x H x H.





Multi Agent Deep Learning



Multi Agent Reinforcement Learning

Simulation

http://busoniu.net/repository.php
http://busoniu.net/files/repository/2010-08-04_marl-1.3.zip

startuprl.m
alg_gui.m
rrgwdemo.m