Machine Learning

- Machine Learning is concerned with the construction and study of systems that learn from data (alternative: improves performance based on experience)
- In general three categories – depending on the information available
  - **Unsupervised Learning**: Cases for learning come without information about the solution; → clustering based on a similarity measure
  - **Supervised Learning**:
    Each case also contains its solution. Approaches generalize / generate knowledge for handling **previously unseen cases**.

    →Decision Tree Learning

    →Neural Networks (later in the lecture on connectionist approaches)
  - **Reinforcement Learning**: The system receives a numeric feedback describing how good its suggestion for solving a case was, the system learns what are good solutions to cases

Clustering

- CBR is not machine learning → it is not producing / elicitating new knowledge from given examples, it just uses a given case base. (Although adding new example may be coined as learning)
- Machine Learning approach with most relation to CBR is **Clustering**:

  Clustering is the process of finding groups such that similar objects are put into the same group (and different objects into different groups).
- Used for
  - To establish prototypes, or detect outliers (for e.g. testing case bases)
  - To simplify data for further analysis/learning.
  - To visualize data (in conjunction with dimensionality reduction) → analysis of case bases
- **Clustering is not Classification**
- There is no ”right” or ”wrong” clustering; different clustering criteria may show different things about the data

K-Means Clustering

- One of the most commonly used clustering algorithms
- Assumes that cases (objects, instances...) to be clustered are n-dimensional vectors x_i
- Uses a **distance measure** between instances
- Goal is to partition the data into **K disjoint subsets** → exclusive

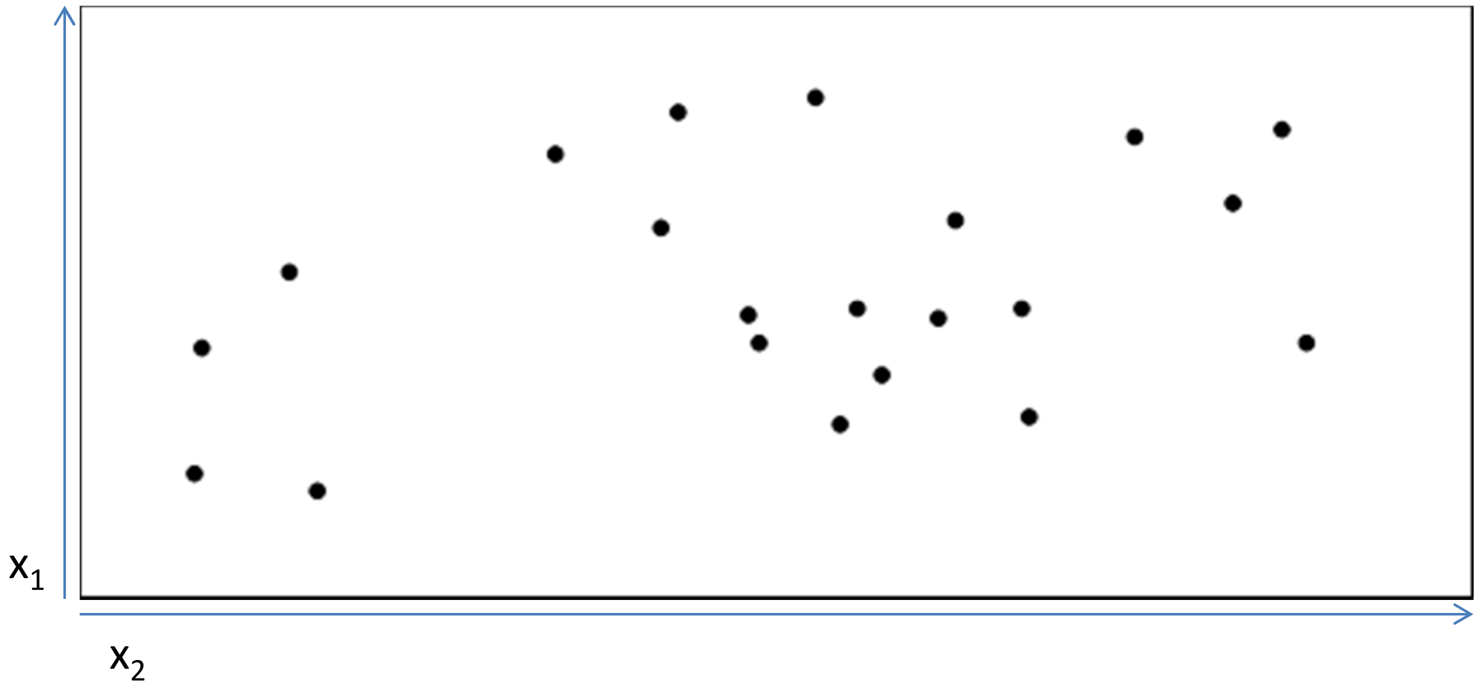
= Ertel-Book, Section 8.7
Slides additionally based on material from a Machine Learning lecture by Doina Precup

K-Means Clustering

- **Inputs**:
  - A set of n-dimensional vectors $\{x_{i,} x_{i, \ldots,} x_m\}$
  - K, the number of desired clusters
- **Output**:
  - Mapping between instance vectors to K clusters: C:{1,...,m}$\rightarrow${1,...,K}
- **Principle**
  1. Initialize C randomly
  2. Repeat:
     a) Compute the mean of all instances in each cluster (”centroid” of the cluster)
     b) Reassign each instance to the cluster with the closest centroid

     until the mapping does not change any more

Example



How many clusters do you see?

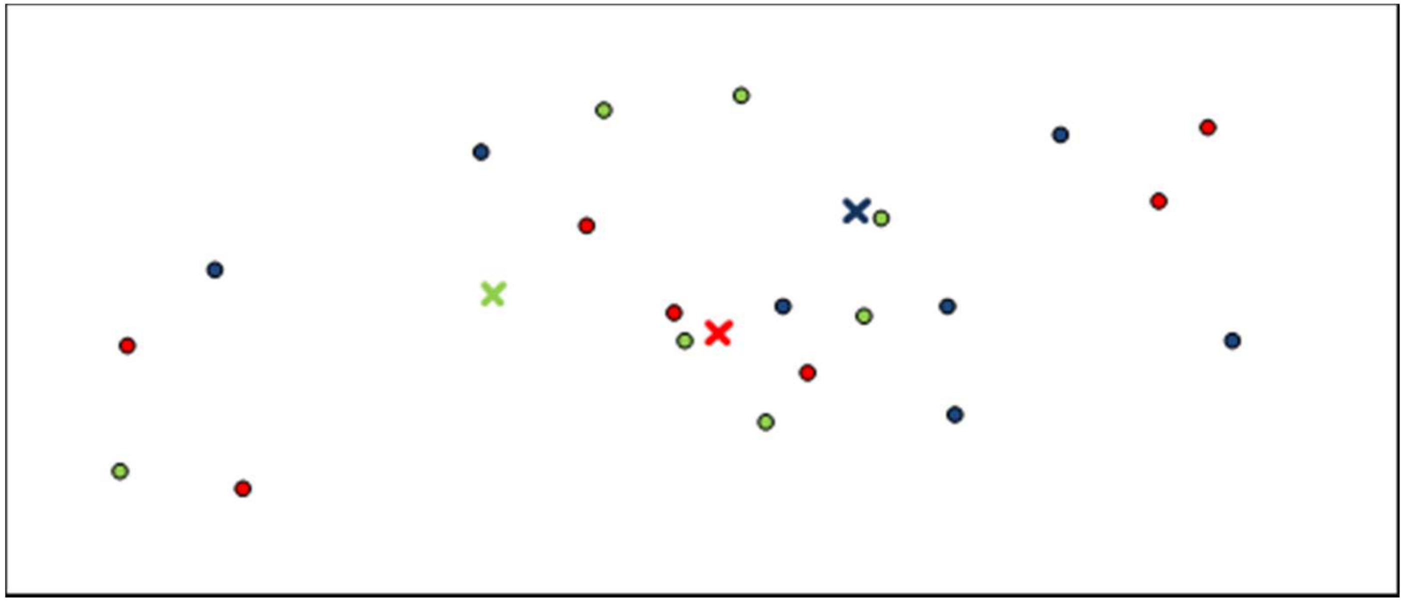
Distance Metrics

- Examples for k-means clustering not for arbitrary attribute values, but mostly for examples only with real-valued attribute sets
- Often distance measures as similarity measures:
  - Minkowski Distance between two numeric attribute vectors $x=\{x_1,\ldots,x_n\}$ and $y=\{y_1,\ldots,y_n\}$

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

    p=1: Manhattan Distance; p=2: Euklidean Distance
- **All similarity measures defined for CBR can also be used.**
- Distance metric is the only knowledge used for clustering , no classifying information on to which group an instance shall belong.
  → ”unsupervised learning”

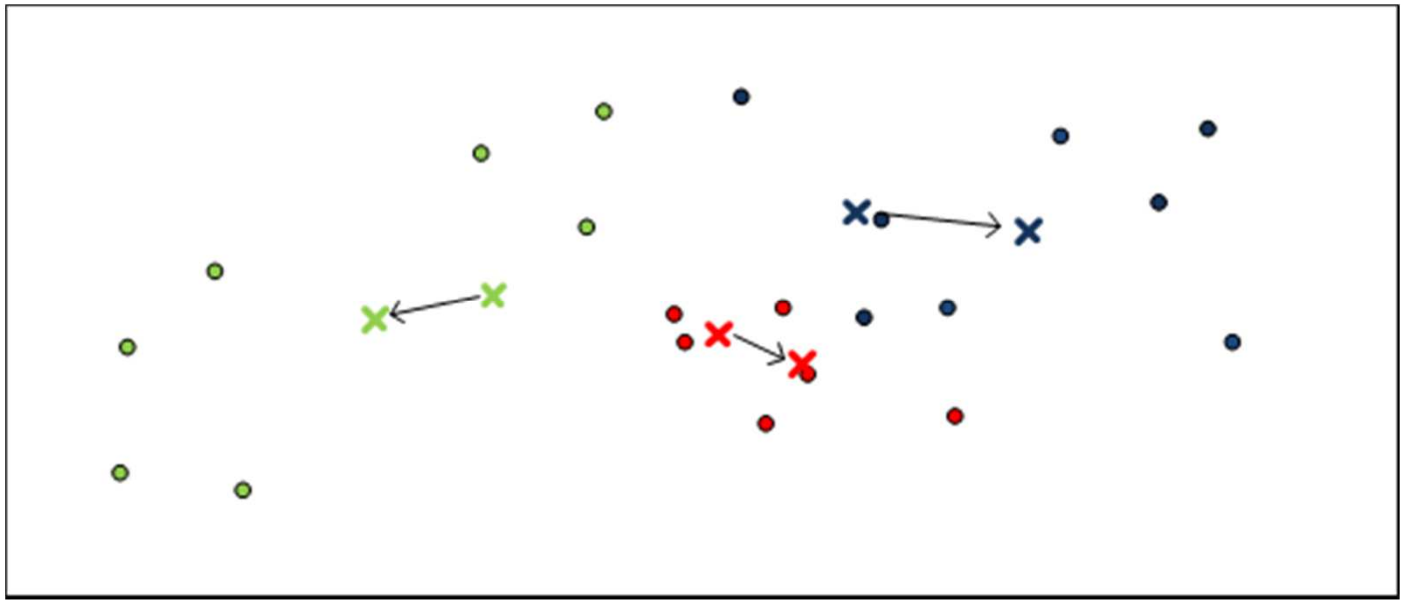
Random mapping with k=3



Centroids are defined as the **mean** of all points belonging to the same cluster:

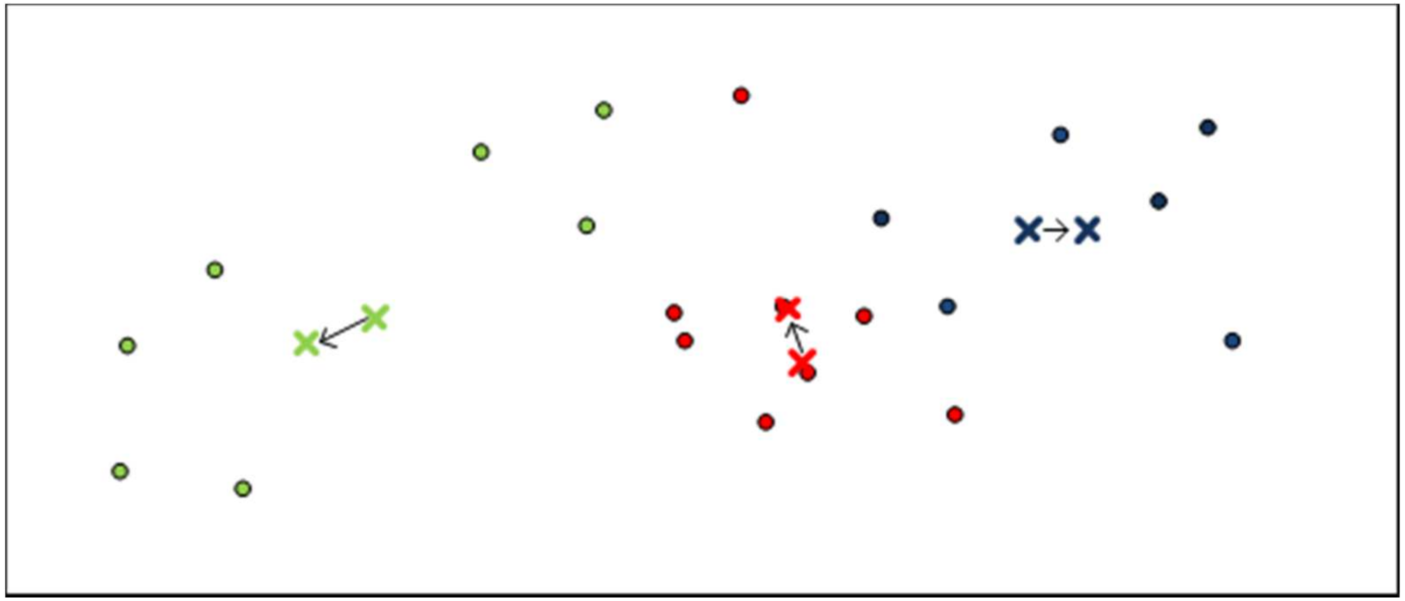
$$\mu = \frac{1}{l}\sum_{i=1}^{l} x_i$$

Update



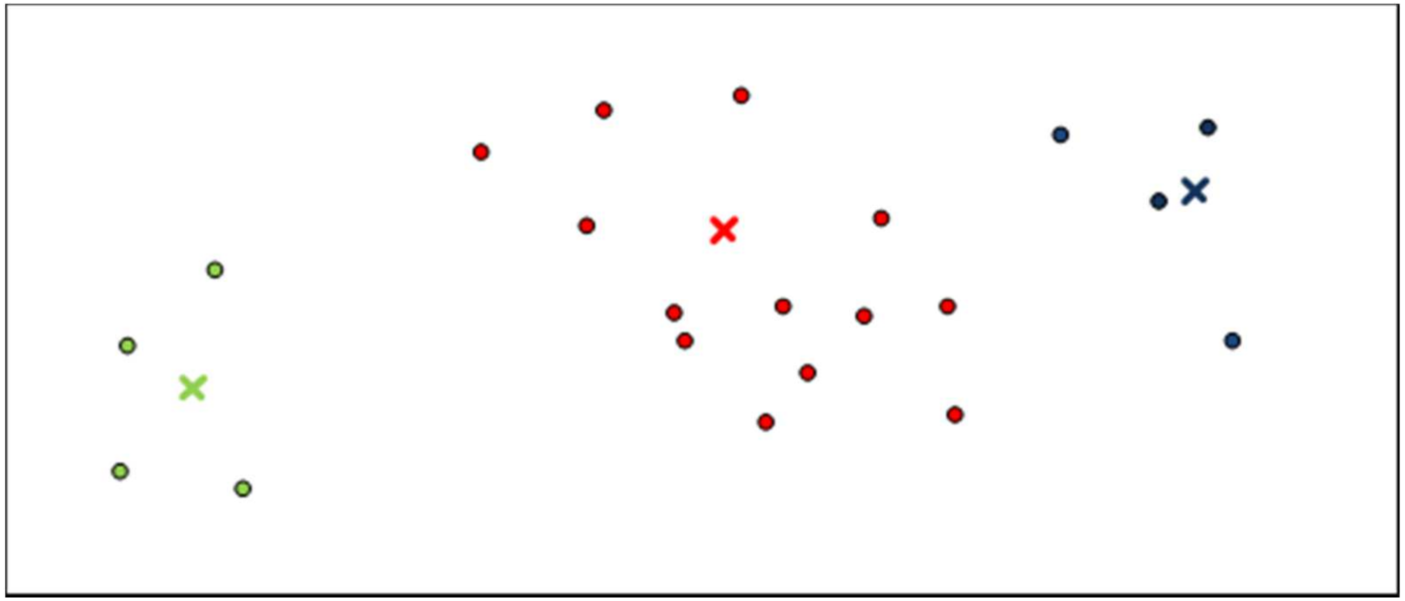
Re-assignment of each example to nearest centroid and update of centriods

And again



... Until none of the instances is re-assigned.

Final assignment



- K-means always terminates: finite number of assignments, every change decreases the overall sum of distances between instances and their centroids
- It finds the **local** optimum, depending on the initial assignment of instances to clusters

Initial Assignments

- A random assignment is unbiased, but may initialize with too many centroids near the centroid of all instances
- A better idea might be the following for placing centroids:
  - Place the first centroid on a randomly choosen data point
  - Place the second on a data point as far away as possible from the first one
  - Place the i-th centroid as far away as possible from centroids 1 to (i-1)
- Alternative:
  Re-run the clustering with a new random assignment and check wether the overall sum of distances for every instance to its centroid is lower

How to choose ”K”?

- Difficult problem → Try and Error?

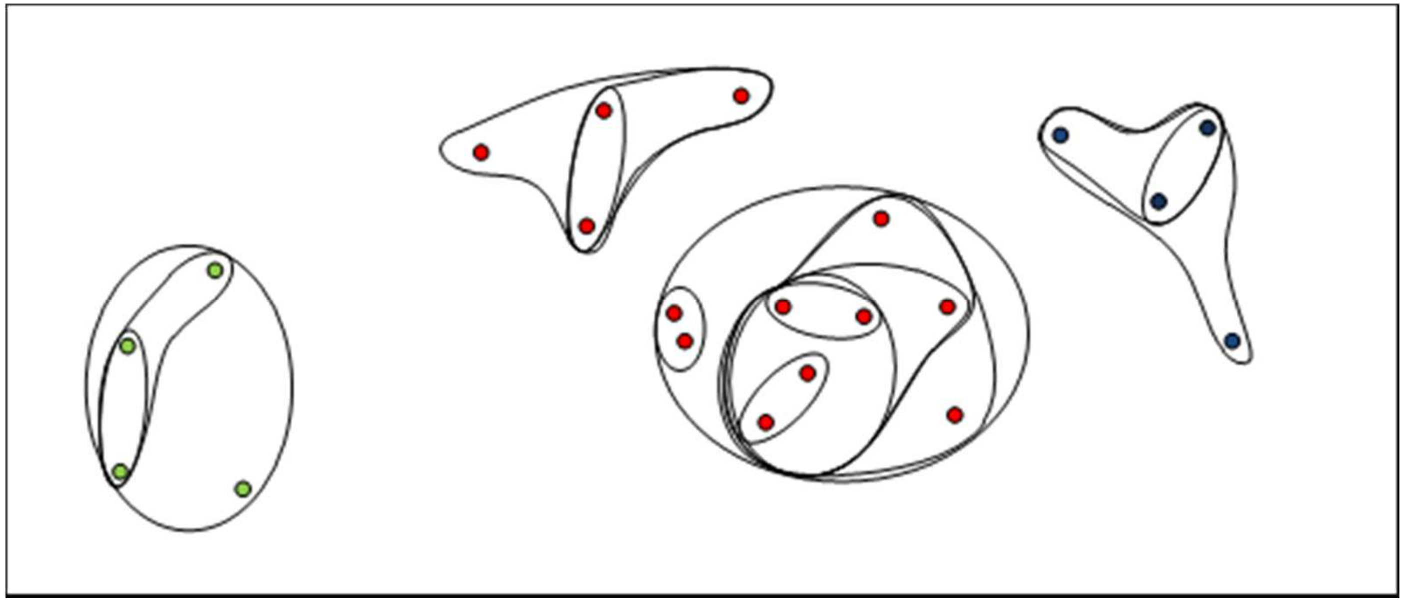
- Ideas for improvements:
  - Delete clusters with too few instances
  - Split clusters with too many instances
  - Add extra clusters for ”outliers”
  - Use hiearchical methods (with suitable termination criterium)

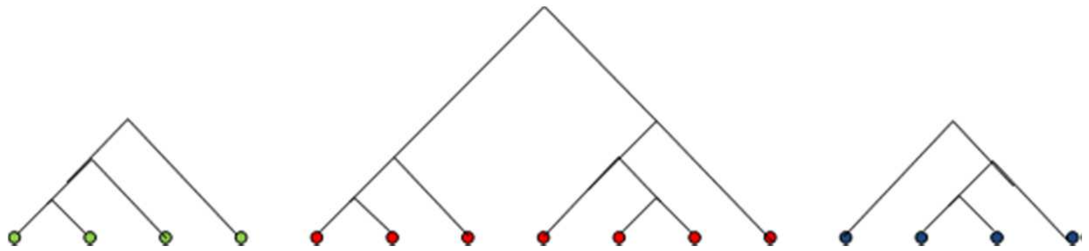
- Choose a scoring function for the overall clustering and optimize its value
  - Sum of pairwise distances within a cluster → minimize
  - Minimum distance between two instances in different cluster → maximize
  - Etc.

Hierarchical Clustering

- Different methods, top-down or bottom-up.
- Bottom-up method:
  - Start with n clusters, each consisting of one instance
  - Repeat combining the two clusters with the minimum distance to a new cluster, until a termination criterium has been reached
- Different termination criteria possible: particular number of clusters, or maximum distance within the cluster...

Example





Distance between Clusters

- Distance of clusters is not equal to distance of instances

- Candidates (with different effects)
  - Distance of the two nearest instances (larger structures)
  - Distance of the two farest instances (compact structures)
  - Distance between cluster centroids (in between)

Supervised Learning

- The cases for learning also contain solution.
- Approaches generalize / generate knowledge for handling **previously unseen cases**.

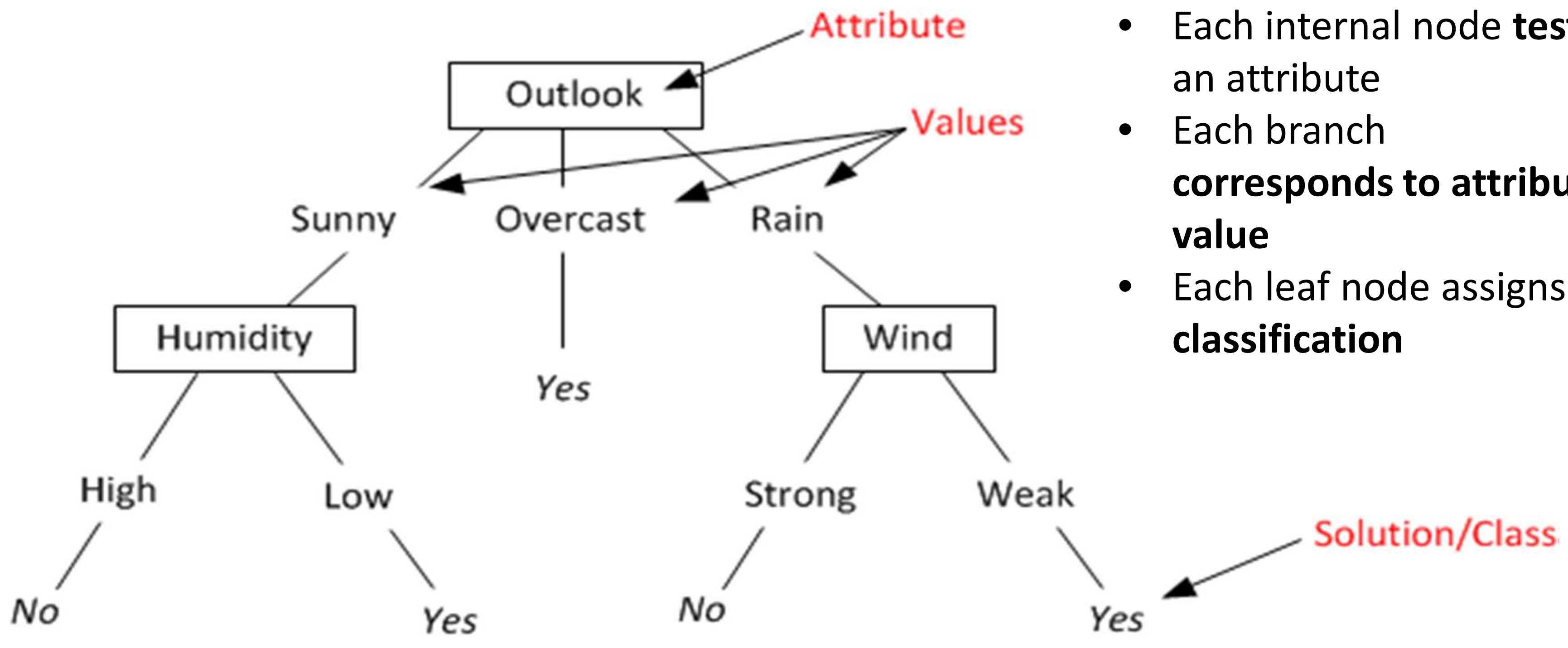
  → Decision Tree Learning

  → Neural Networks

  → ...

- Supervised learning often for **classification tasks**:
  - Given is a (finite) set of classes: {c1,...cn}
    and a description of an entity (object, case...), select the class it belongs to
  - In learning for supervised learning one generates a function **from a data base of classified entities** with which a previously unseen entity can be (”properly”) classified

Decision Tree for Playing Tennis



- Each internal node **tests** an attribute
- Each branch **corresponds to attribute value**
- Each leaf node assigns a **classification**

Inspiration from lecture slides to Tom Mitchell: Machine Learning, McGraw Hill 1997

Decision Trees in general

- Using a decision tree:
  - **Follow the path given by attribute-value pairs of the problem description**
  - If value for attribute is unknown: use statistics/weighted probabilities
- When Decision Trees?
  - Instances describable using attribute-value pairs
  - Target (class) function has discrete value
  - Possibly noisy training data
  - Disjunctive solutions may be required
  - Interpret-ablilty by humans is needed
- Application areas:
  - Simple decision making (e.g for non-player characters in games)
  - Data mining
  - ...

Decision Trees and Rules

- **Each path between root and result can be represented as rule:**
  if-part: conjunctions of the attribute tests on the nodes
  then-part: classification/result of the branch

- So, the tennis tree can represented by

  if outlook=sunny $\wedge$ humidity=low $\Rightarrow$ Yes, play tennis
  if outlook=overcast $\Rightarrow$ Yes
  if outlook=rain $\wedge$ wind=weak $\Rightarrow$ Yes

  if outlook=rain $\wedge$ wind=strong $\Rightarrow$ No
  if outlook=sunny $\wedge$ humidity= high $\Rightarrow$ No

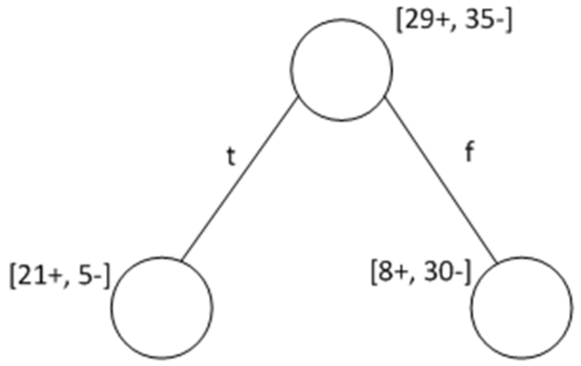
Learning Decision Trees from Examples

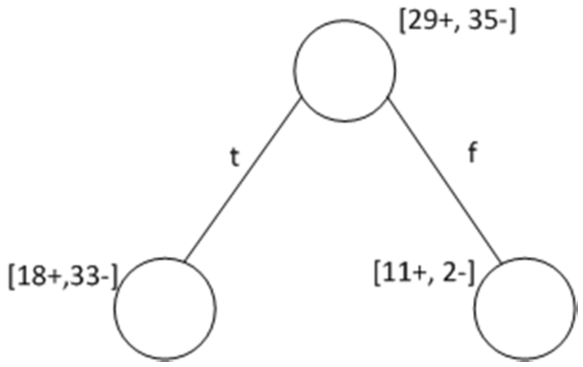
- Prominent learning techniques are based on **top-down induction**
- Learning from examples/cases

Outlook	Temperatur	Humidity	Wind	Play (Class)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Weak	No
Overcast	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes
Rainy	Cool	Normal	Strong	No
Overcast	Cool	Strong	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rainy	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Principle

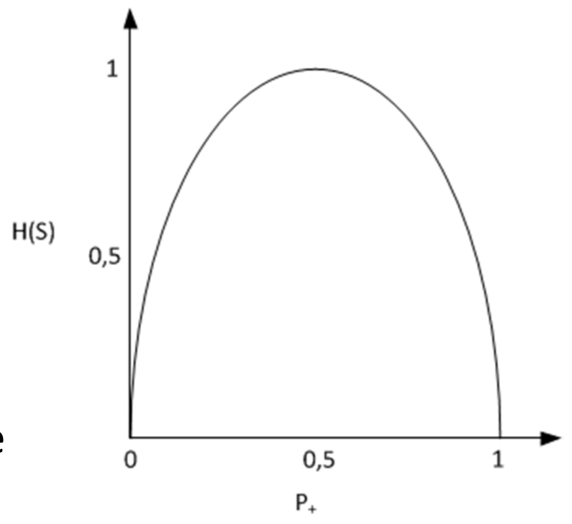
- Main Loop:
  - A ← ”best” decision attribute for next node
  - For each value v in the range of A, create a new subtree for value v and check for all examples with A=v:
    - if all examples are classified with the same class or no other attribute are left, create leaf node
    - otherwise (recursively) create subtree with this subset of examples and rest of attributes
- Which attribute is the ”best”?





→ use the attribute that gives us the most information for classification

Information Gain



- **Entropy** H(S) is a measure for **impurity** of a set of examples S: the more mixed the classes are in S, the higher H(S):

  $H(S) = - p_- * \log(p_-) – p_+ * \log(p_+)$ for binary decision attribute

  or $H(S) = \sum_{i=1}^{n} -p_i \log_2 p_i$ for attributes with n values
  (p_i is the proportion of examples with value i for the *class* attribute)
- Entropy of S as the number of bits needed to encode the class of any member of S (assuming the code with shortest length)
- Information Gain (S, A) = expected reduction in entropy due to sorting on A

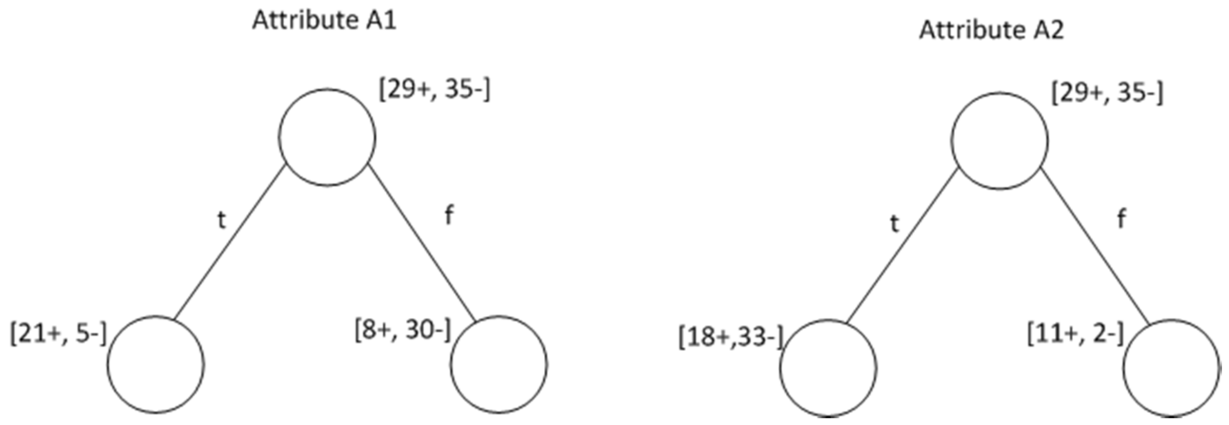
$$InformationGain\ (S, A) = H(S) - H(S|A)$$

with $H(S|A) = \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$

(S_v = Set of examples with value *v* for A: $H(S_v)$ = H(S|A=v))

→ Calculate Information Gain for each attribute, select the one with the highest value.

Information Gain



$$H(S) = \sum_{i=1}^{n} -p_i \log_2 p_i$$
$$IG(S,A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

- Calculate H(S) for the overall set:

$$H(S) = -\frac{29}{29+35}\log_2\frac{29}{29+35} - \frac{35}{29+35}\log_2\frac{35}{29+35} = 0{,}993651$$

- Attribute A1:

$H(S|A_1 = t) = -\frac{5}{21+5}\log_2\frac{5}{21+5} - \frac{21}{21+5}\log_2\frac{21}{21+5}$ = 0,706274

$H(S|A_1 = f) = -\frac{8}{30+8}\log_2\frac{8}{30+8} - \frac{30}{30+8}\log_2\frac{30}{30+8}$ = 0,742488

$H(S|A_1) = \frac{21+5}{29+35}H(S|A_1 = t) + \frac{8+30}{29+35}H(S|A_1 = f)$ = 0,72776

$\rightarrow$ $IG(S,A_1) = H(S) - H(S|A_1) = 0{,}265875$
$\rightarrow$ $IG(S,A_2) = H(S) - H(S|A_2) = 0{,}121432$

→ Attribute with higher Information Gain is A1

ID3 Algorithm Pseudo Code

```
Function ID3(ExamplesTable, Attributes)
•   If all examples in ExamplesTable have same class,
    return leaf-tree, with label = class
•   If Attributes list is empty,
    return leaf-tree, with label = most common class in ExamplesTable
•   In all other cases:
    – A ← attribute in Attributes with highest information gain
    – Create new Tree T with decision attribute A,
      for each possible value v of A
        • Examples_v ← subset of Examples with A==v
        • If Examples_v is empty: add a leaf node with label = most common
          class in Examples
        • Else add ID3(Examples_v , Attributes − {A}) as a new subtree at the
          branch with A = v
•   return Tree T
```

Pseudo Code: Calculate Information Gain

```
Function InformationGain(Examples, A)
"A is the attribute whichs information gain shall be calculated,
S is the decision attribute"
•   HS = CalculateEntropy(Examples, S)
•   HA = 0
•   for each possible value vi of A
            • E_v ← subset of Examples with v for A
            • HA ← HA + (|E_v|/|Examples|)*CalculateEntropy(E_v, S)
•   return HS – HA


For function CalculateEntropy uses formula given above
```

Example Base

Outlook	Temperatur	Humidity	Wind	Play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Weak	No
Overcast	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Normal	Weak	Yes
Rainy	Cool	Normal	Strong	No
Overcast	Cool	Strong	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rainy	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rainy	Mild	High	Strong	No

What is the Attribute with highest IG?

- Start with H(S) – S is the binary decision attribute **Play**
- We have 14 cases of which 5 are Play=Yes and 9 are Play=False: [5+,9-]
  → $H(S) = H(Play) = -\frac{5}{14}\log_2\frac{5}{14} - \frac{9}{14}\log_2\frac{9}{14} = 0{,}941$
- Next: For each attribute in {Outlook, Temperatur, Humidity, Wind} determine the conditional entropy H(S|A)
- **Outlook** has range {sunny,overcast, rainy}

$$\begin{aligned} &H(S|Outlook) \\ &\quad = \frac{\#sunny}{\#all} H(S|Outlook = sunny) \\ &\quad + \frac{\#overcast}{\#all} H(S|Outlook = overcast) \\ &\quad + \frac{\#rainy}{\#all} H(S|Outlook = rainy) \end{aligned}$$

What is the Attribute with highest IG?

- Outlook = sunny

Outlook	Temperatur	Humidity	Wind	Play
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Weak	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

- $H(S|Outlook = sunny) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}$=0,971
- Similarly:
  H(S|Outlook = rainy) with 3 Yes and 2 No → 0,971
  H(S|Outlook = overcast) with 4 yes → 0

→ H(S|Outlook) = 0,694

→ Gain(S, Outlook) = H(S)-H(S|Outlook) = 0,247

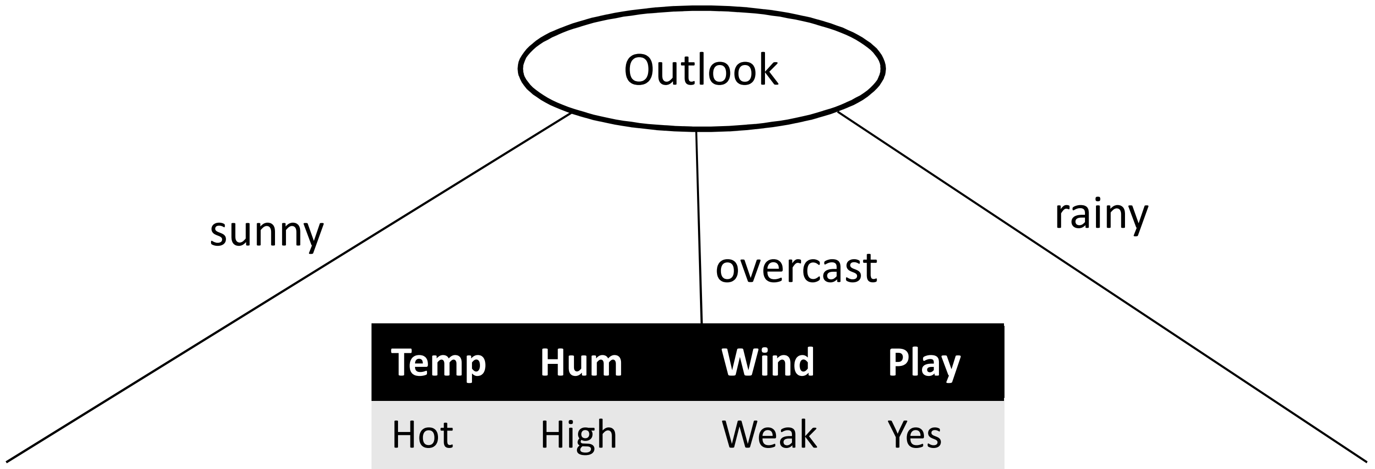
What is the Attribute with highest IG?

- **Similar calculations** for the other attributes Temperatur, Humidity, Wind
  Gain(S, Temperature) = 0,029
  Gain(S, Humidity) = 0,151
  Gain(S, Wind) = 0,048

→ With Gain(S, Outlook) = 0,247 **Outlook is the Attribute** with the highest IG

→ Generate tree with outlook as the attribute and branches for the values

Root Attribute Outlook



sunny

Temp	Hum	Wind	Play
Hot	High	Weak	No
Hot	High	Weak	No
Mild	High	Weak	No
Cool	Normal	Weak	Yes
Mild	Normal	Strong	Yes

overcast

Temp	Hum	Wind	Play
Hot	High	Weak	Yes
Cool	Strong	Strong	Yes
Mild	High	Strong	Yes
Hot	Normal	Weak	Yes

rainy

Temp	Hum	Wind	Play
Mild	High	Weak	Yes
Cool	Normal	Weak	Yes
Cool	Normal	Strong	No
Mild	Normal	Weak	Yes
Mild	High	Strong	No

Recursion → Subtree: Outlook=sunny

- Restrict the examples to consider to the table at branch
- Remaining attributes: {Temperature, Humidity, Wind}
- Reduced to table:
  $H(S_{sunny}) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}$=0,971

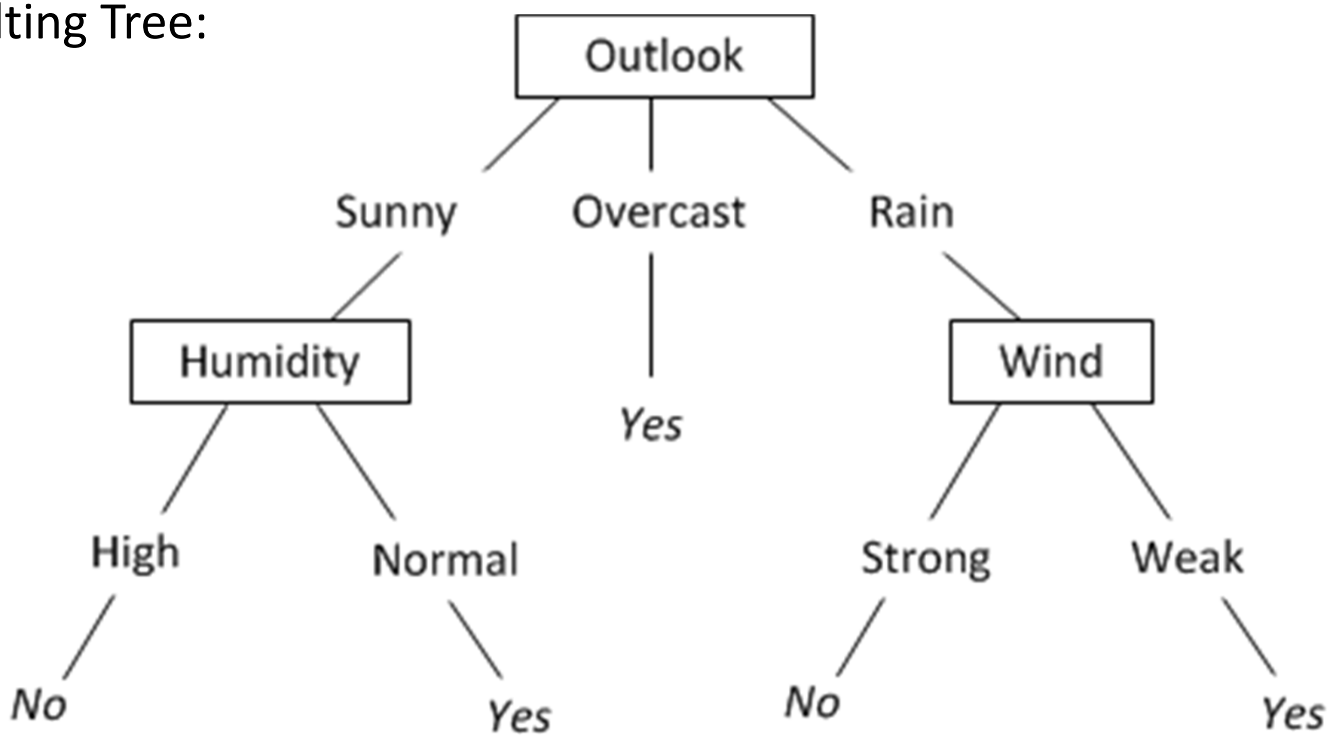
Temp	Hum	Wind	Play
Hot	High	Weak	No
Hot	High	Weak	No
Mild	High	Weak	No
Cool	Normal	Weak	Yes
Mild	Normal	Strong	Yes

- Consider Temperature:
  H($S_{sunny}, Temp = Hot) = 0$
  H($S_{sunny}, Temp = Mild) = 1$
  H($S_{sunny}, Temp = Cool) = 0$
  →*H(*$S_{sunny}, Temp) = \frac{2}{5}0 + \frac{2}{5}1 + \frac{1}{5}0 = 0.4$ → *Gain*($S_{sunny}, Temp$)=0,571
- Similarly:
  Gain($S_{sunny}, Humidity$)=0,971
  Gain($S_{sunny}, Wind$)=0,322

→ Humidity is the attribute with the highest gain → next decision node

And so on...

- Analogous calculations for outlook=rainy
- For the branch outlook=overcast we do not do any further calculations as all cases lead to the same decision
- Resulting Tree:



Attributes with Many Values

- Problem:
  - **If there is an attribute with many values, Information Gain will select it**
  - But, something like Date=Dec_12_2013 or Names as an attribute does not support generalization.
- Solution:
  - (ignore an attribute for which each case has a different value)
  - Replace the usage of simple information gain for selecting the best attribute with Ratio of information gain:

$$GainRatio(S,A) = \frac{IG(S,A)}{SplitInformation(S,A)}$$ with

$$SplitInformation(S,A) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

with S_i is subset of S for which A has value v_i

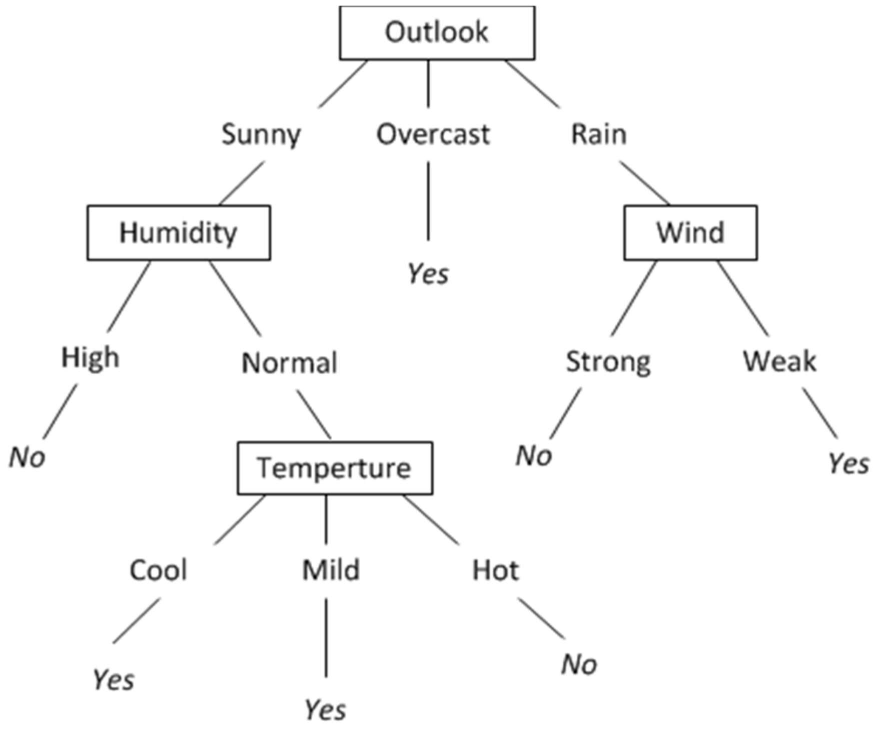
What is a good Decision Tree?

- For n binary attributes → 2^n possible decision trees
- A good decision tree is the **minimal one to do proper decision** (”Occam’s Razor”: prefer the shortest hypothesis that fits the data)
- The shallower a decision tree, the more compact → The more compact, the less information is needed to make a decision.
- Too much compactness results in **underfitting**. → the decision tree is too simple
- **Overfitting** occurs when the tree uses more branching than necessary (noise) → Pruning

Overfitting

- What if we would add noisy example
  Sunny, Hot, Normal, Strong with play=No to the examples
- What would be the effect on the tree?

Temp	Wind	Play
Cool	Weak	Yes
Mild	Strong	Yes
Hot	Strong	No



Avoid Overfitting

- How can we avoid overfitting?
  - Stop growing when data split is not statistically significant
  - Grow full tree and post-prune

- How to measure quality of a tree?
  - Measure performance over training data (How many cases are correctly classified)
  - Measure performance over separate validation test set

  → Optimize tree for a combination of size and miss-classifications

Machine Learning

- Learning is essential for unknown environments, i.e., when designer lacks omniscience
- Learning is useful as a system construction method, i.e., expose the agent to reality rather than trying to write it down
- Learning modifies the agent's decision mechanisms to improve performance
- Machine Learning often for classification problems:
  - Is a message spam? Does the patient have this disease? Does the engine operate normally?
  - Learning to recognize complex signal
  - Learning to select the “right” action
  - Learn about similar entries

→ Ertel Book, Chapter 10

Remember: Different Learning Approaches

- ✓ **Supervised Learning**: Learning from cases with given solutions
- ✓ **Unsupervised Learning**: Learning without given solutions

- **NOW: Reinforcement Learning**: Learning based reward feedback
  - Modelbased RL
  - Q-Learning

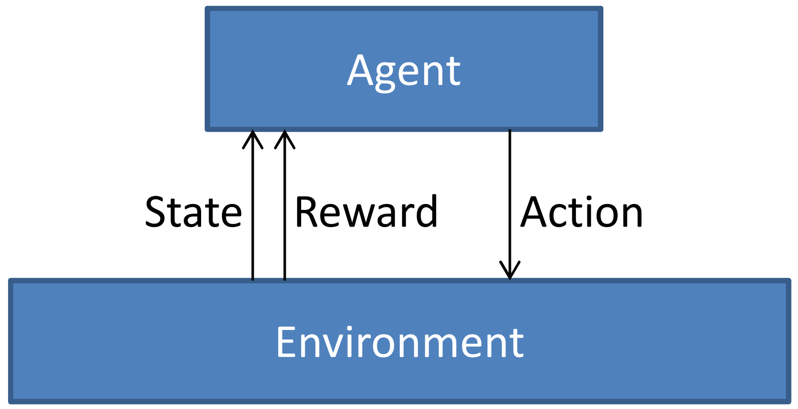
Reinforcement Learning

- See Pavlov’s dogs
- Sequential decision problem
- Learning by trial-and-error which actions are good
- Robotics, not programmable actions are learnt

- Setup: learning agent (robot)
  – the agent has a set of *sensors* to observe the *state* of its environment
  – And a set of *actions* it can perform to alter its state
  – task is to learn a control strategy, or *policy*, for choosing actions that achieve its goals

- **Basic assumption:** goals can be defined by a *reward function* that assigns a numerical value to each distinct action the agent may perform from each distinct state → There is NO presentation of correct Input-Solution pairs!

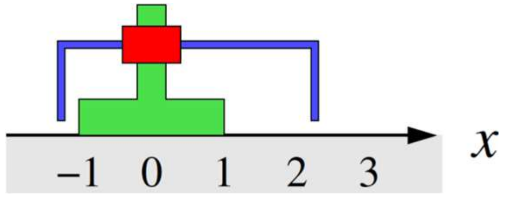
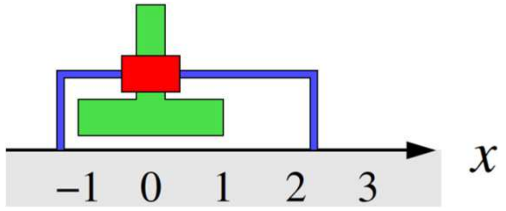
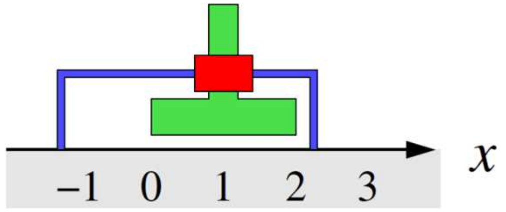
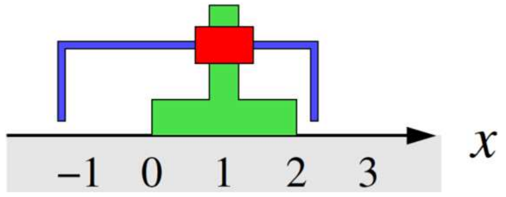
That means...



$$\begin{tikzcd} s_o \arrow[r, "a_o", "r_o"'] & s_1 \arrow[r, "a_1", "r_1"'] & s_2 \arrow[r, "a_2", "r_2"'] & s_3 \arrow[r, "a_3", "r_3"'] & {} \end{tikzcd}$$

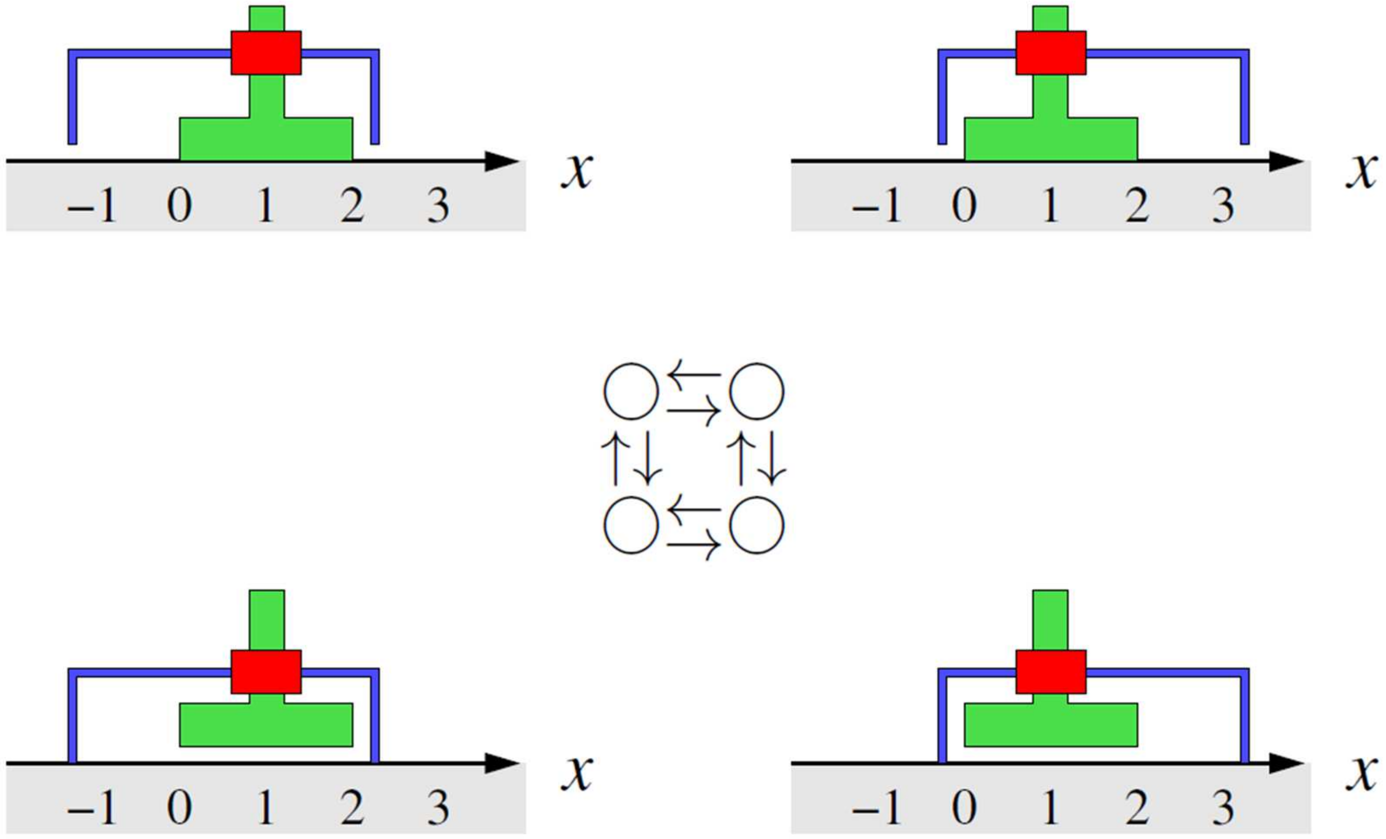
- The optimal policy for each state returns the action that maximes the discounted cummulative reward:
- **Policy must be complete**: In each state, the agent knows what action to select

Policy for Walking Robot

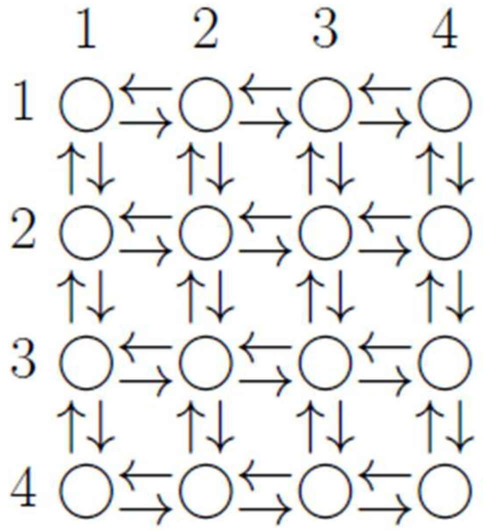
Robot	State gx	State gy	Reward	Action
	Up	Left	0	Right
	Up	Right	0	Down
	Down	Right	0	Left
	Down	Left	1	Up

Ertel Book, p. 259

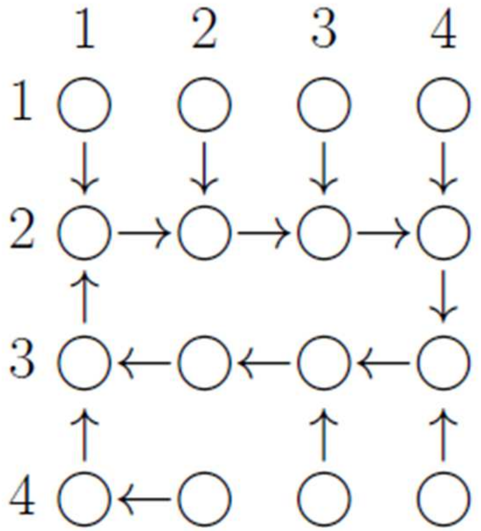
State Space



State Space and Optimal Policy



A state space with 4x4 states



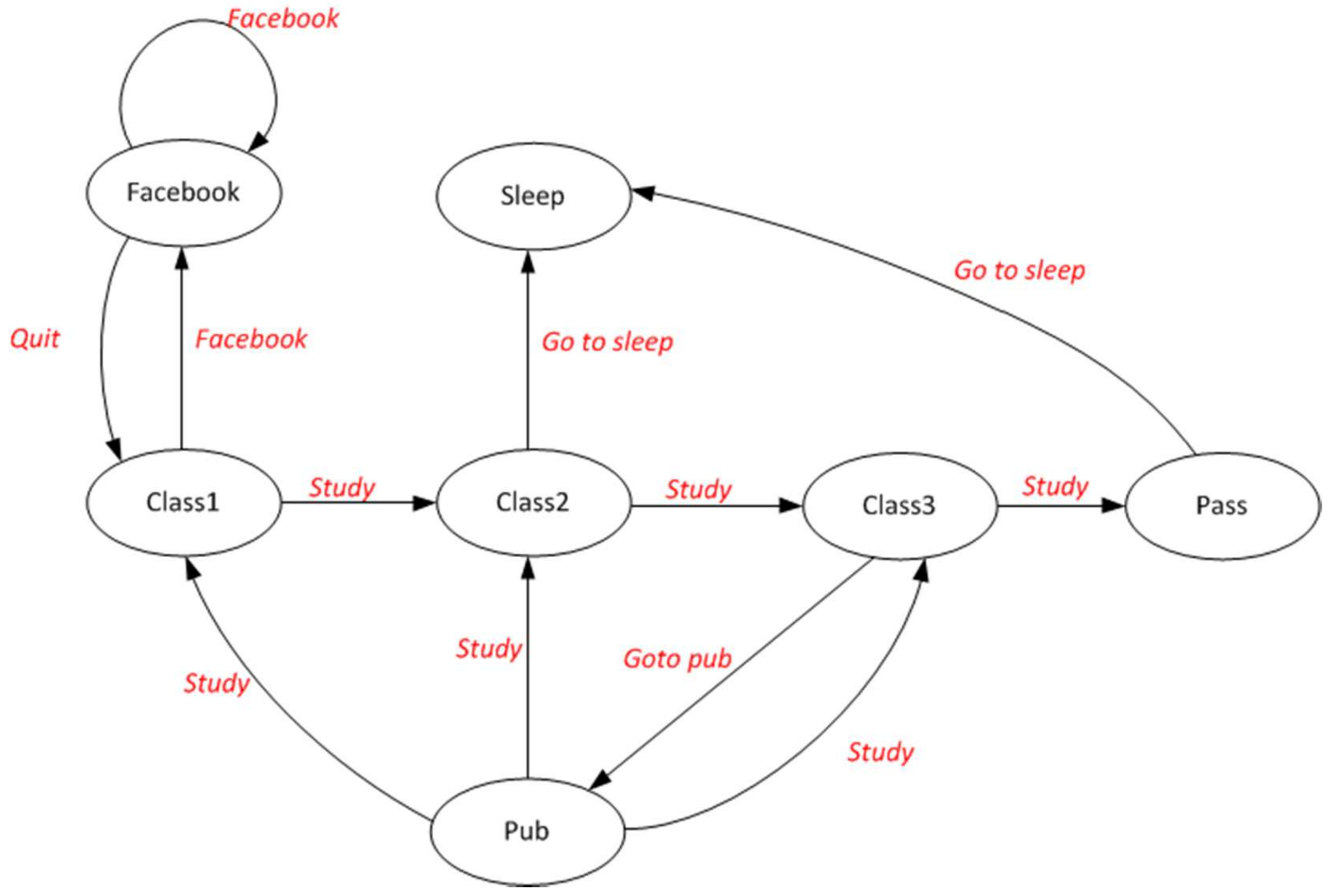
A optimal strategy, giving the ”best” action in each state.

Markov Decision Process (MDP)

- the agent can perceive a set **S** of **distinct states** of its environment and has a set **A** of **actions** that it can perform
- at each discrete time step t, the agent senses the **current state** s_t, chooses a **current action** a_t and performs it
- the environment responds by returning a **reward** $r_t = r(s_t, a_t)$ and by producing the **successor state** $s_{t+1} = \delta(s_t, a_t)$;
  δ may be nondeterministic.
- the functions r and δ are part of the environment and not necessarily known to the agent

- **Markov Property**: the functions $r(s_t, a_t)$ and $\delta(s_t, a_t)$ depend only on the current state and action

(deterministic) Example



The student markov chain, adapted from Lecture of David Silver, University College London)

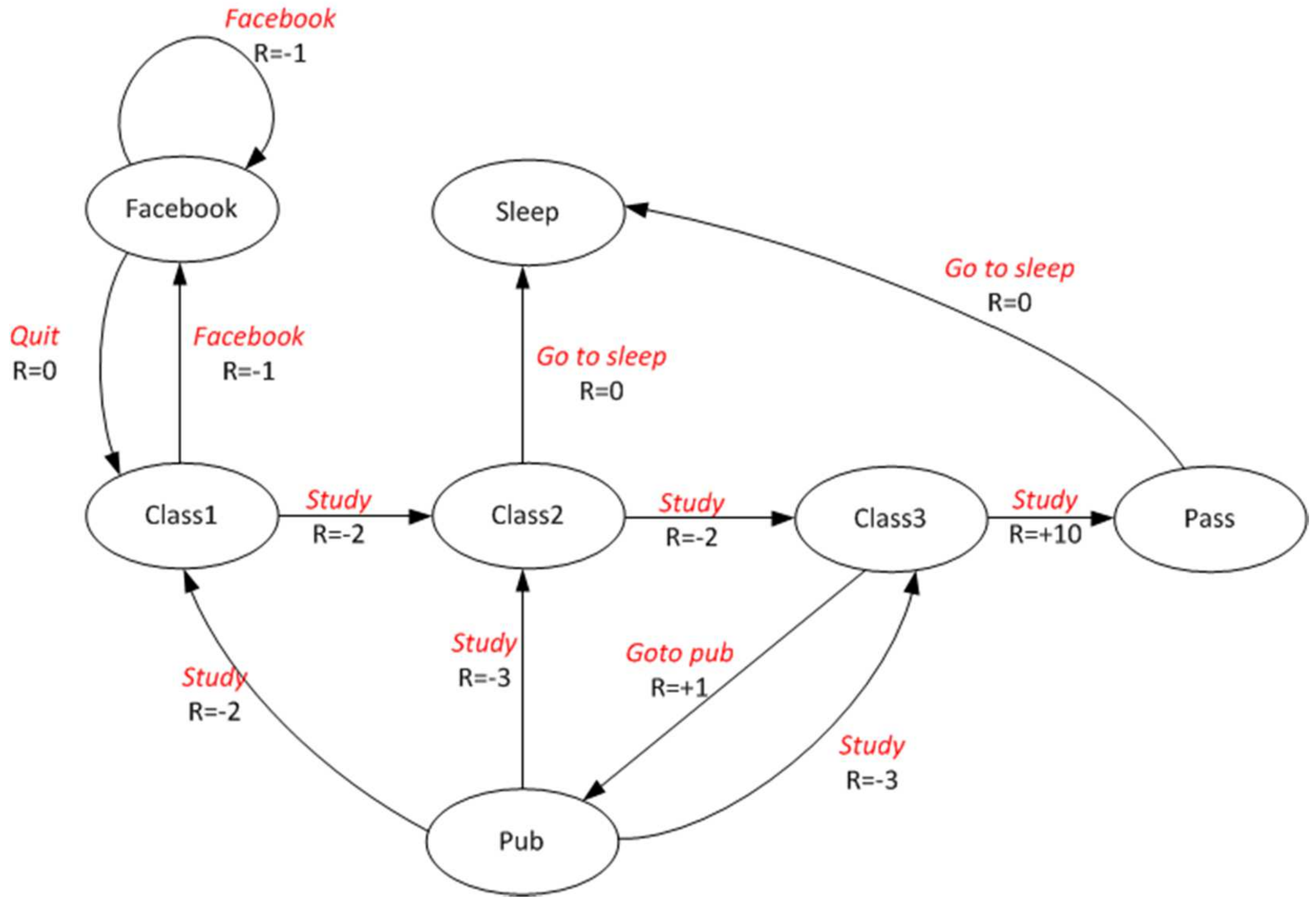
Reward

- r_t is a scalar feedback signal for an action done in a state
- Indicates how well agent is doing at step t

→ **agent wants to optimize sum of reward over time**

- Examples of reward:
  - Fly stunt maneuvers with a helicopter:
    + reward for following a given trajectory
    - reward for crashing
  - Defeat the world champion in Backgammon
    + reward for winning a game
    - reward for loosing a game
  - Manage an investment portfolio
    + reward for each $ on bank account / win

In the Example



The student markov chain, adapted from Lecture of David Silver, University College London)

Learning Task

- Task is to learn a **policy**: π: $S \rightarrow A$
  That means a function that selects for every perceived state the best action. **Which one is "best"?**
- the policy π that produces the greatest possible sum of reward over time (**discounted cumulative reward**):

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

  where $V^{\pi}(s_t)$ is the cumulative value achieved
  by following a policy π from an arbitrary initial state s_t
- r_{t+1} is generated by repeatedly using the policy π.
- Discount factor γ ($0 \leq \gamma < 1$) is a constant that determines the relative value of delayed versus immediate reward: if γ=0, the agent just pays attention to the reward achievable in the next time step. $\gamma \rightarrow 1$, only rewards in the future are relevant

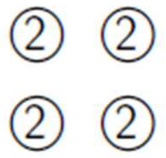
Optimal Policy

- A policy π* is optimal if:

$$V^{\pi*}(s) \geq V^{\pi}(s)$$

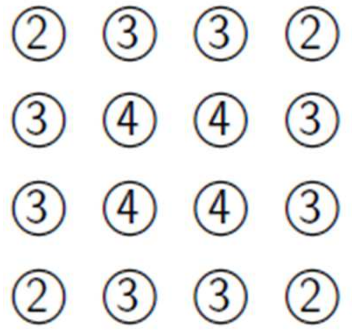
  for all states s

- **There are many policies**:
  e.g. 4 states with 2 actions each → 2^4 = 16 policies

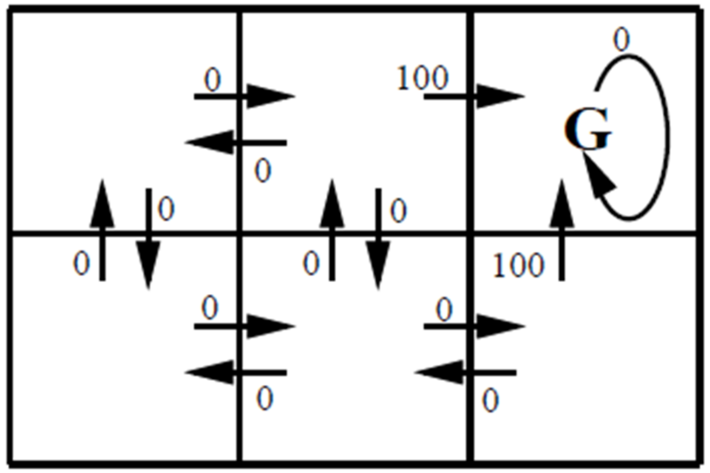


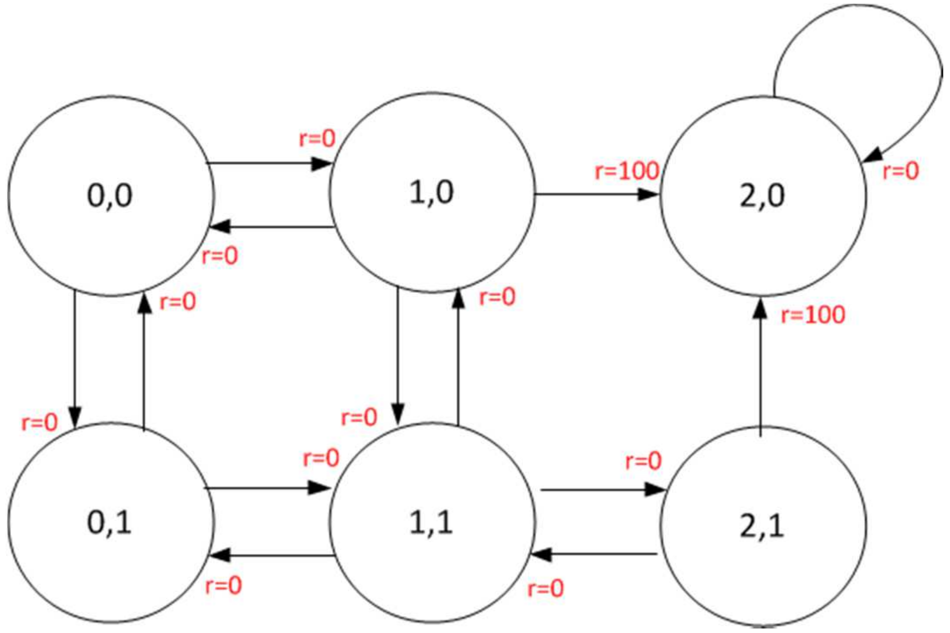
  e.g. 16 states with 2 actions each:

  Corner nodes: 2^4
  Nodes at edges: 3^8
  Inner nodles: 4^4
  $\approx 2{,}7*10^7$



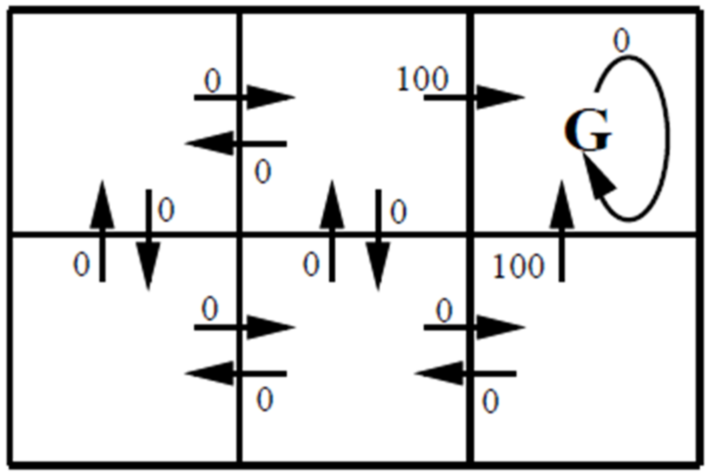
Cumulative Discounted Reward Values?

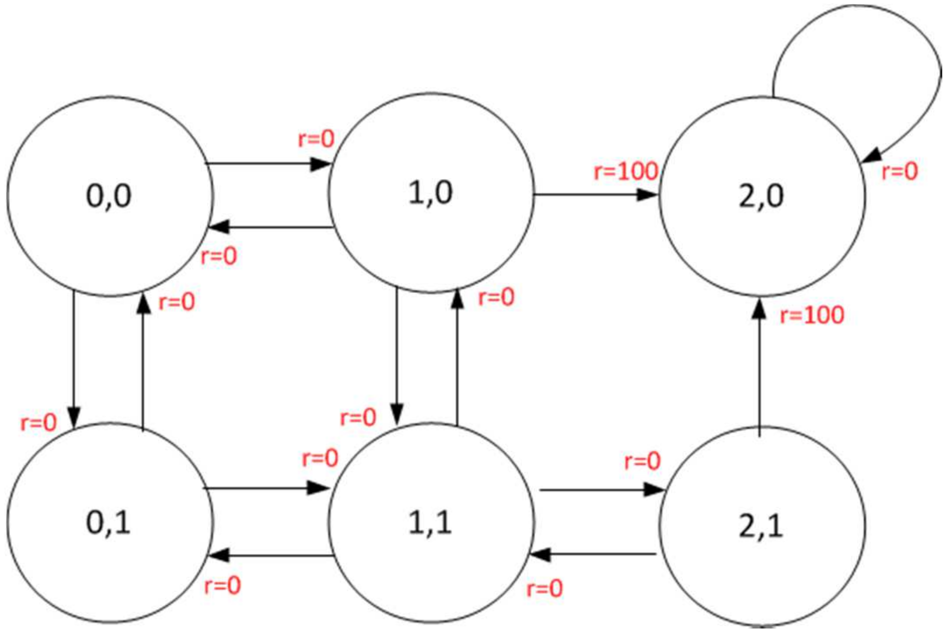




- Simple Grid World
  - Squares ≈ state, location s
  - Arrows ≈ possible action a with annotated reward r(s,a)
  - G ≈ Goal state; absorbing state → only when passing over to goal, there is positive feedback.
  - Discount factor $\gamma = 0.9$

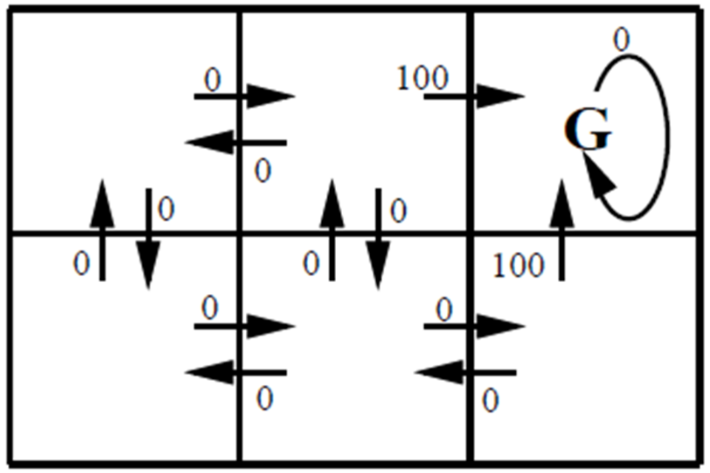
Cumulative Discounted Reward Values?

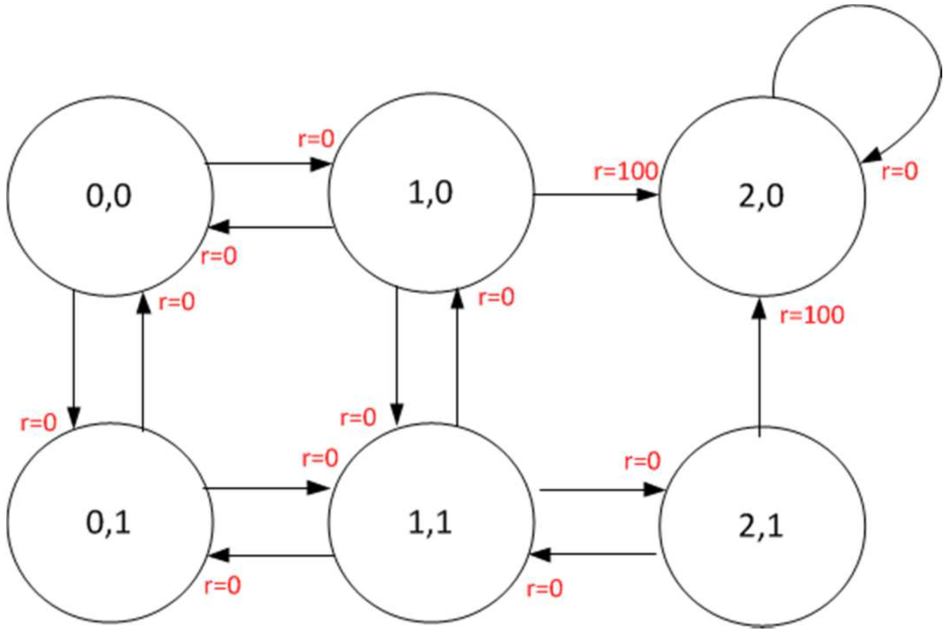




- **V*(s) is sum of discounted rewards over infinite future!**)
- Consider the bottom-center state:
  - V*=90, because π* selects the ”move right” and ”move up” actions: V* = 0 + $\gamma \cdot 100 + \gamma^2 \cdot 0 + \ldots = 90$

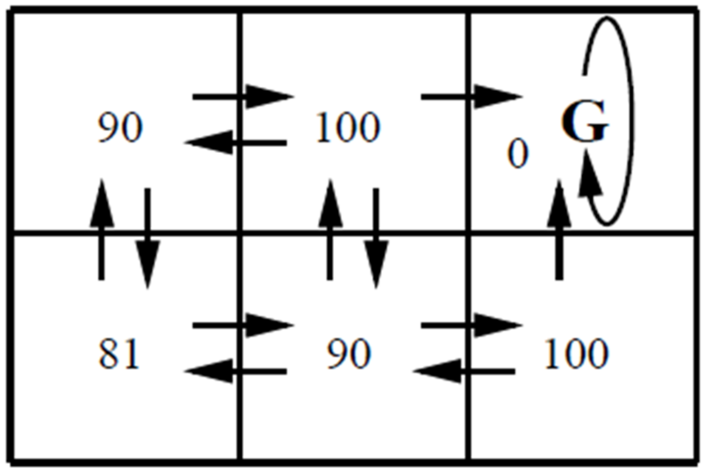
Cumulative Discounted Reward Values?





- **V*(s) is sum of discounted rewards over infinite future!**)
- Consider the bottom-right state:
  - V* = 100, because π* (the optimal policy) selects ”move up” action that receives a reward of 100
  - Then the agent will stay in G and receive no further awards: $V^* = 100 + \gamma \cdot 0 + \gamma^2 \cdot 0 + \ldots = 100$

Cumulative Discounted Reward Values



This is the result with V* assigned to every cell.
An agent could now check successor states of potential moves/action and select the action with the highest value.
→ **Thus an agent can aquire the optimal policy by determining V***
(provided the agent has perfect knowledge of the immediate reward function r and the state transition function δ

In General

- Optimal strategy V^* maximizes:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

- r_t depend on s_t and a_t: $r(s_t,a_t)$:

$$V^*(s_t) = \max_{a_t} r(s_t, a_t) + \gamma \max_{a_{t+1}, a_{t+2}, \ldots}(\quad r(s_{t+1}, a_{t+1}) + \gamma r(s_{t+2}, a_{t+2}) + \ldots) =$$
$$\max_{a_t} r(s_t, a_t) + \gamma V^*(s_{t+1})$$

- Bellman Equation (fixpoint equation)

$$V^*(s) = \max_{a} r(s, a) + \gamma V^*(\delta(s, a))$$

→ This leads to the ”Value Iteration” algorithm

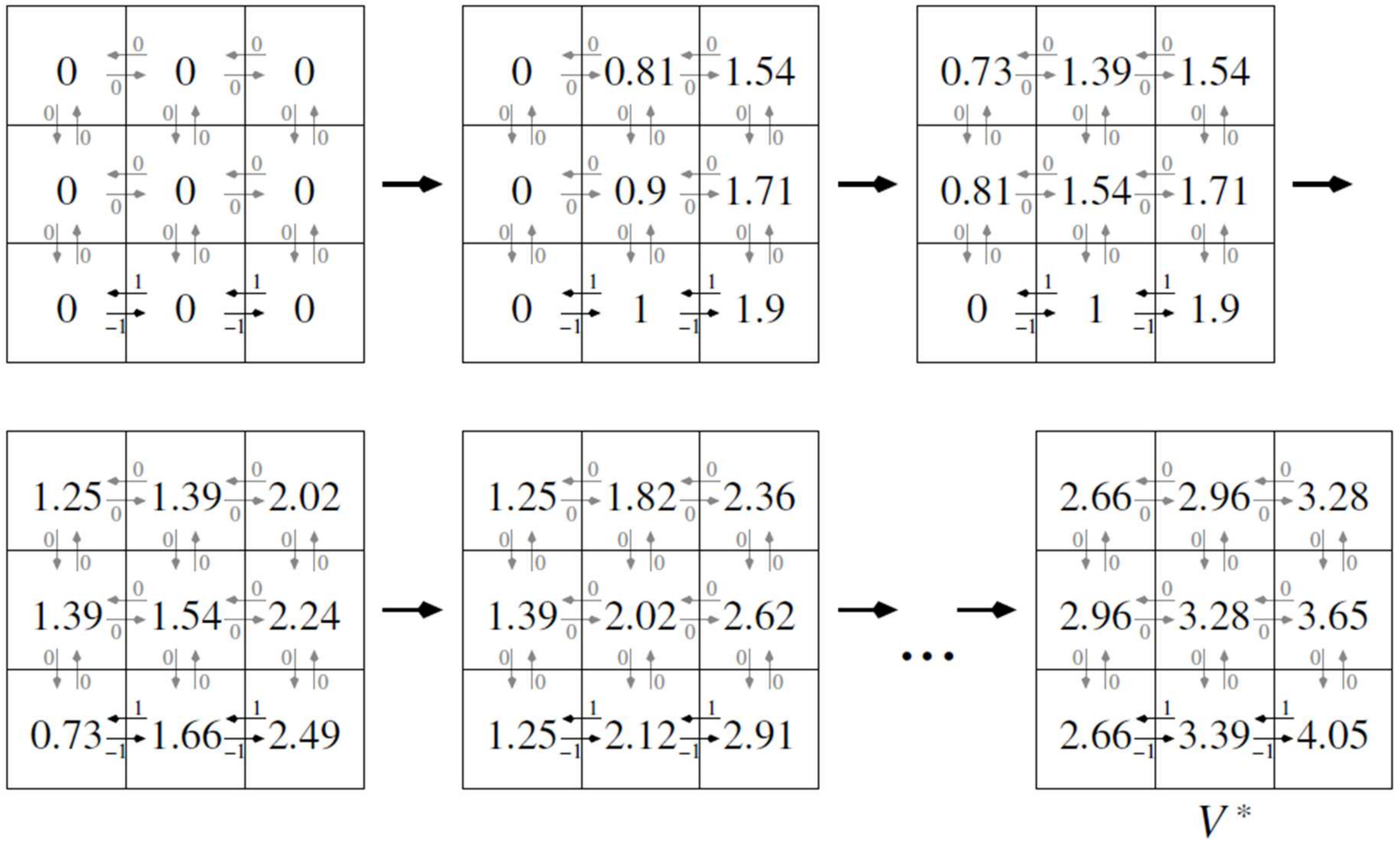
Value Iteration

```
Function **valueIteration()**
    For all s in S:
        $\hat{V}(s) = 0$
    Repeat:
        For all s in S:
            $\hat{V}(s) = max_a\left[r(s,a) + \gamma\hat{V}\left(\delta(s,a)\right)\right]$
    Until $\hat{V}(s)$ converged
    (= differences between two iterations are smaller than threshold)
```

Barto&Sutton: Value iteration converges to V*

For using that algorithm we MUST know r and δ (and transitions need to be deterministic)

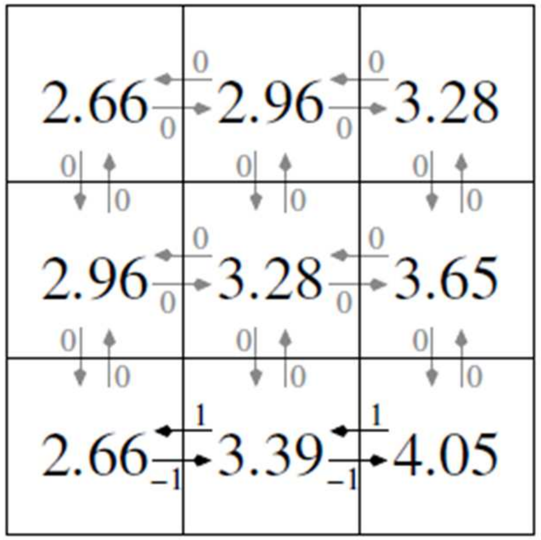
Example



γ=0-9

Selecting the next action

- Select the action that gives maximum reward plus V* of the resulting state:
- $\pi^*(s) = \underset{a}{\operatorname{argmax}}(r(s,a) + \gamma V^*(\delta(s,a))$



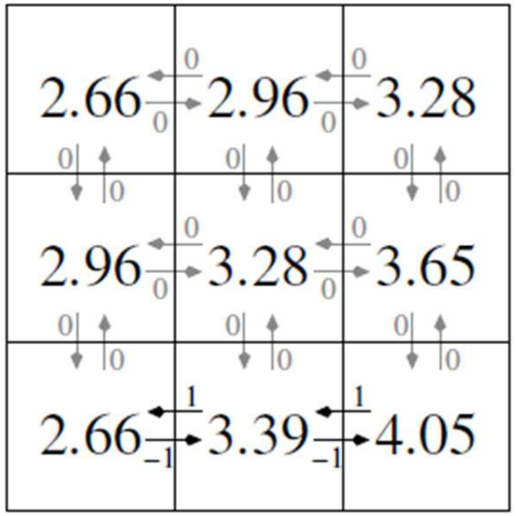
- That means e.g. in state (2,3):

$$\pi^*(2,3) =$$

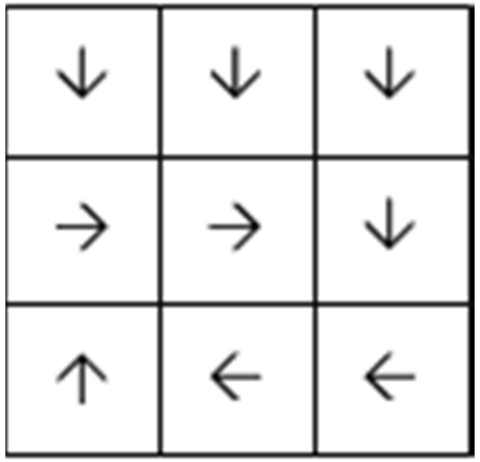
$$\underset{left,up,right}{\operatorname{argmax}} \ (1 + 0.9 * 2{,}66,\ 0 + 0{,}9 * 3{,}28, -1 + 0.9 * 4{,}05) =$$

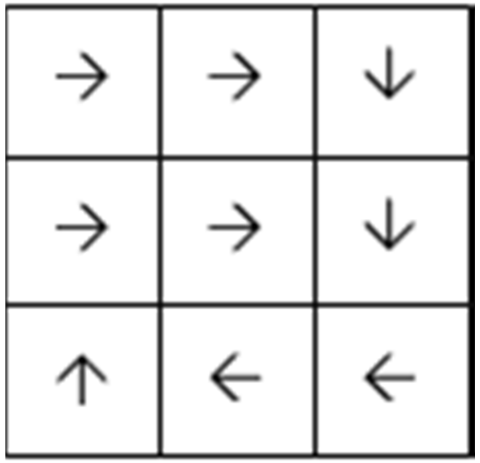
$$\underset{left,up,right}{\operatorname{argmax}} \ (3{,}39, 2{,}65, 2{,}95) = \textit{left}$$

Optimal Policy in the Example

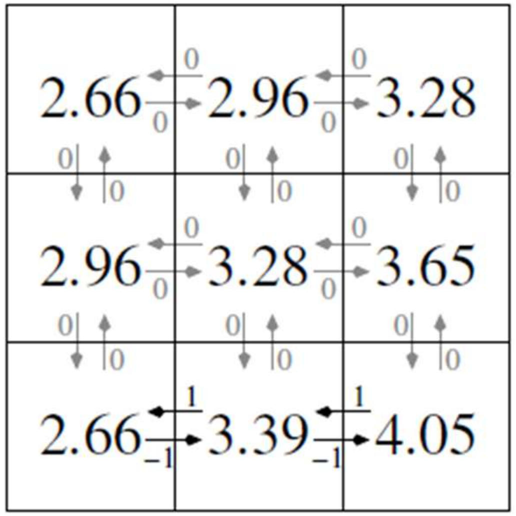


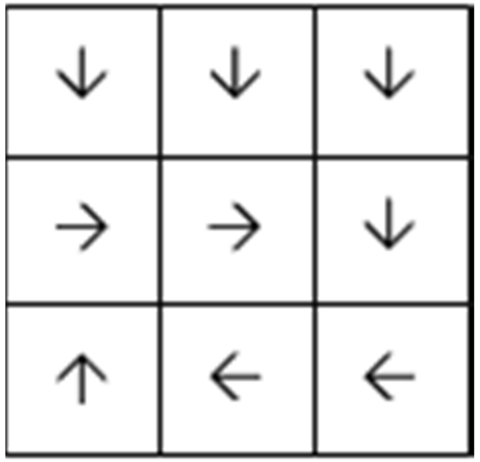
leads to the following two policies

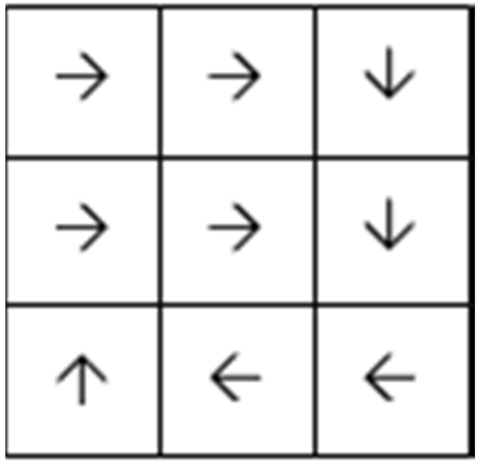




Optimal Policy in the Example







leads to the following two policies

What if we do not know r or δ? but just get feedback when the agent actually tried an action?

Q-Learning

- Instead of using an iterative algorithm to determine the optimal policy, **learn the evaluation function for a state-action pair**
- **This evaluation function is called Q-Function**.
- Q-function *indicates* the maximum discounted reward that can be achieved starting from s and applying action a first:
  $\mathrm{Q}(\mathrm{s},\mathrm{a}) = r(s,a) + \gamma V^*(\delta(s,a))$

→ $\pi^*(s) = argmax_a \mathrm{Q}(\mathrm{s},\mathrm{a})$

→ all information is collected in one value for each (s,a) pair

- **Agent learns Q-function from interaction with environment, no explicit knowledge on r, δ required**
- **Exploration**: In current state, agent randomly selects an action from available ones
- **Exploitation**: In current state, agent considers all available actions, chooses the one that maximizes Q

Q-Learning

- The learner's estimate $\hat{Q}$ is represented by a large table with a separate entry for each state-action pair

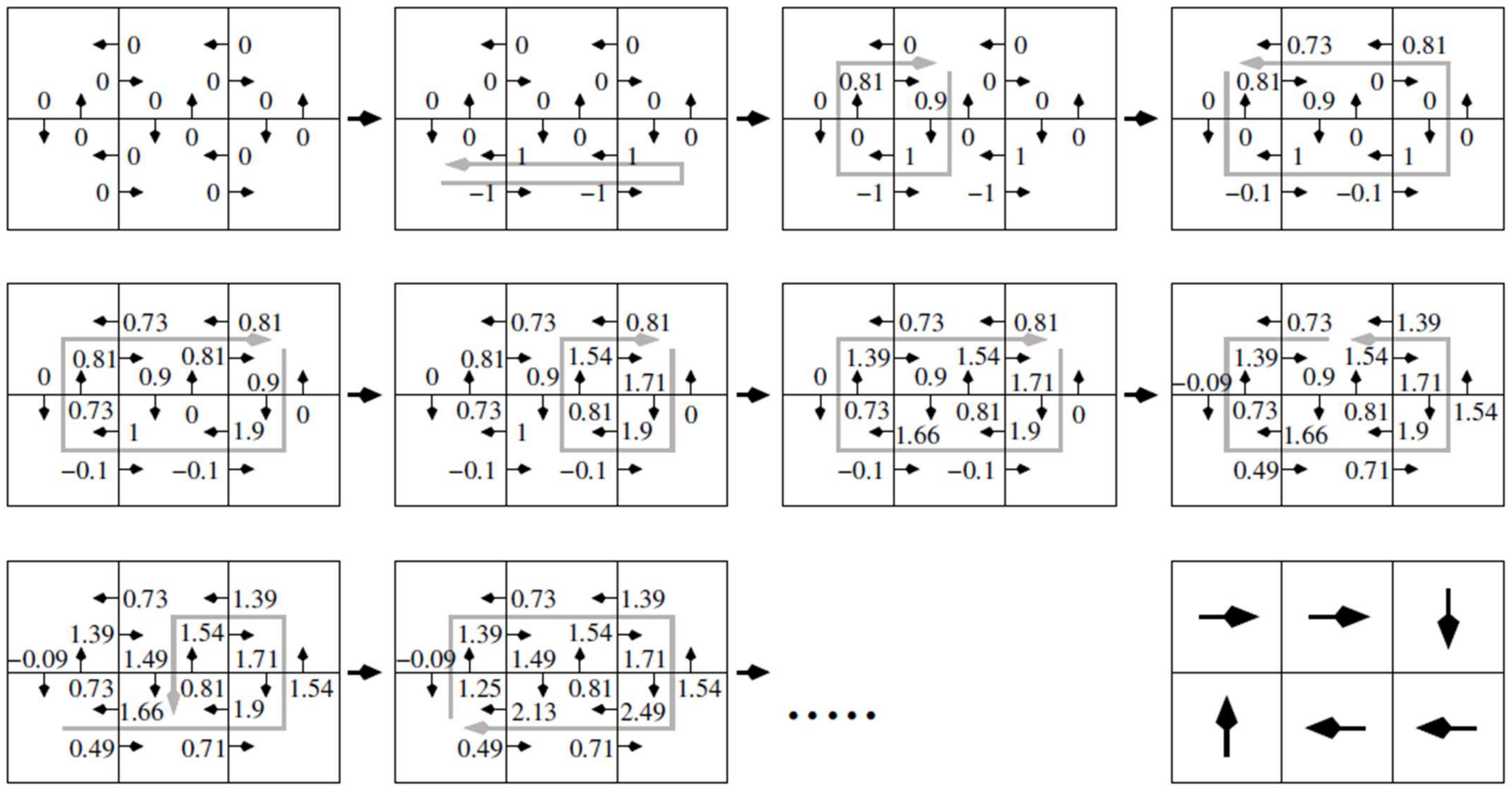
```
Q-Learning Algoritm
 1.  For each s, a initialize the table entry $\hat{Q}(s,a)$ to zero
 2.  Observe current state s
 3.  Loop
       a)  **Select** action a and execute it
       b)  Receive immediate feedback r
       c)  Observe new state s'
       d)  Update $\hat{Q}(s,a)$ : $\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$
       e)  s ← s'
```

Q-Learning

- The learner's estimate $\hat{Q}$ is represented by a large table with a separate entry for each state-action pair

```
Q-Learning Algoritm
1.  For each s, a initialize the table entry $\hat{Q}(s,a)$ to zero
2.  Observe current state s
3.  Loop
                                              Explore/Exploit Strategy
        a)  **Select** action a and execute it
        b)  Receive immediate feedback r
        c)  Observe new state s'
        d)  Update $\hat{Q}(s,a)$ : $\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$
        e)  s ← s'
```

Example (γ=0.9)



Ertel, p. 269

Q-Learning

- Deterministic MPD, infinite number of visits of each states
  → it can be proven that Q-Learning converges

- Convergence is independently of the chosen actions during learning
- Speed of convergence depends on the chosen actions
- If environment is indeterministic → the same action in the same state results in different responses from the environment → convergence garantee is lost

Challenges

- **Reward/Punishment typically delayed:**
  Credit Assignment Problem: how do we know which actions were responsible for the current reward? → include Q-values from more states in the update of the Q-value.
- **How to balance Exploration/Exploitation?**
  - Exploration (try out actions, helps you to improve the model / table): potential techniques:
    - choose a random action
    - choose an action which has not been choosen before
  - Exploitation (take the action that we now know is best)
- **What if the state space is too large to visit all possible states?**

Q-Learning

- Most simple reinforcement learning algorithm
- Model-free approach;
  Currently most popular: SARSA Learning (State-Action-Reward-State-Action)
- Model-based approaches learn the transition function plus the state evaluation.

- Reinforcement Learning has many applications
  - on multiple levels of granularity (→ see Peter Stone and RoboCup)
  - Reward function is super-critical
  - Biggest problem is scalability: size of the Q-Table!
  - If a problem with continuous state has to be solved: for getting a discrete state, usually a Neural Network is used for classifying a continous state and map it to a discrete one.