Part 9:
Machine Learning

Univ.-Prof. Dr. Gerhard Widmer
Department of Computational Perception
Johannes Kepler University Linz

gerhard.widmer@jku.at
http://www.cp.jku.at/people/widmer

Motivation: A Learning Agent

Central Question: Where does an agent's knowledge base come from?

Manual coding, "Knowledge Engineering":

Advantage:

- Can utilise human knowledge and expertise

Problems:

- Easy to miss something
- Fixed knowledge base not adaptive to changes in the world
- There are lots of tasks where we do not have the required knowledge

Automatic Learning:

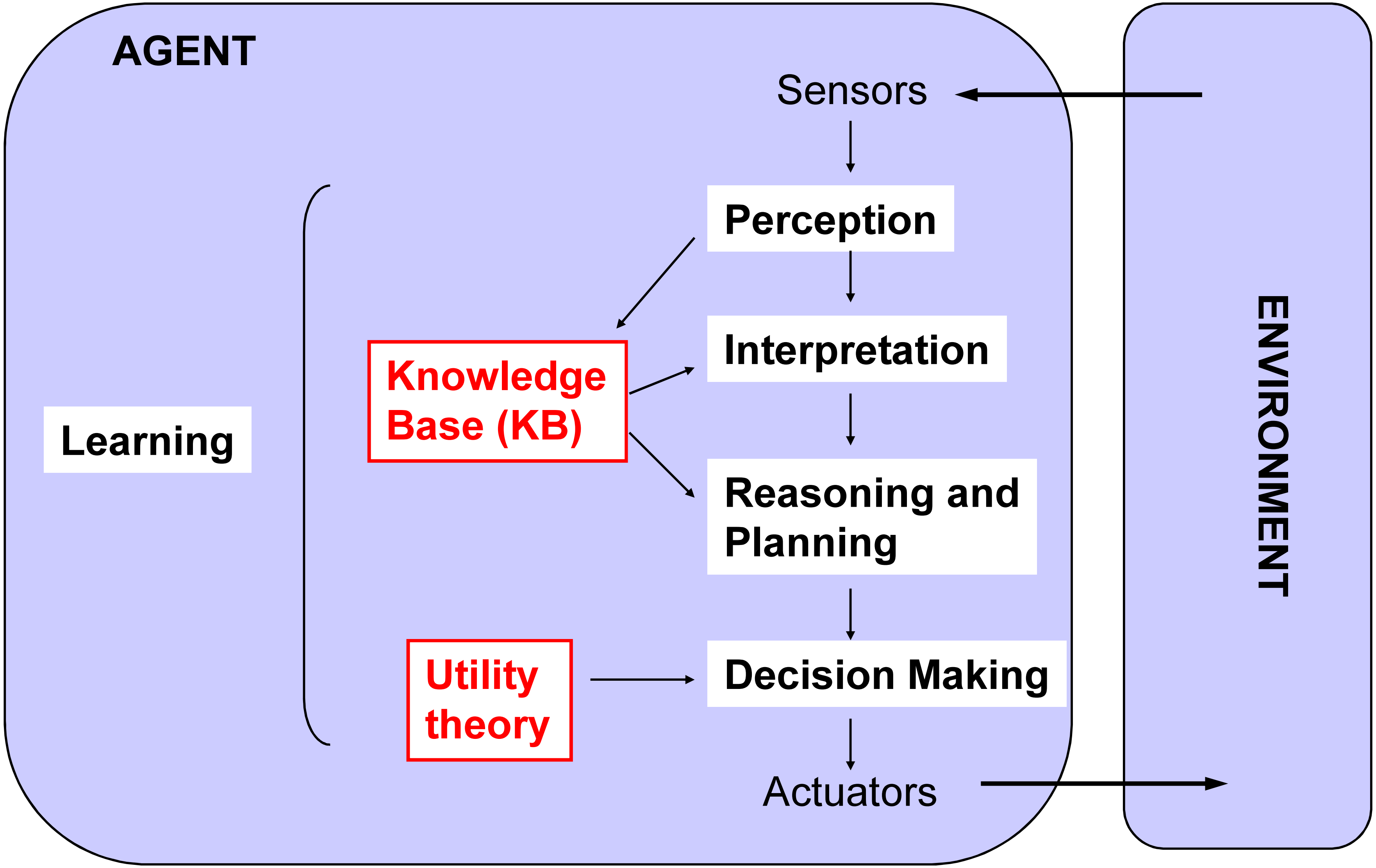
Advantages:

- Can adapt to new environments and changes in the world
- Can fill in missing knowledge that knowledge engineer may have missed

Problems:

- Must make lots of observations / experiences
- Needs reliable feed-back

Knowledge-based Agents: The Full View



What is Learning?

“Learning is making useful changes in our minds.”
(Marvin Minsky, 1985)

“Learning denotes changes in a system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently next time.”
(Herbert Simon, 1983)

“Learning means behaving better as a result of experience.”
(Stuart Russell & Peter Norvig, 1995)

“Learning is constructing or modifying representations of what is being experienced.”
(Ryszard Michalski, 1986)

What is Learning?
Three Fundamental Aspects of Learning

Learning is based on experience:

- A learner needs ***input*** (observations / examples) to learn from
- A learner needs ***feedback*** to evaluate the correctness of its inferences

Learning means (at least) remembering:

- A learner needs a memory and the ability to modify its memory contents

  ➔ Importance of ***knowledge representation***

Learning goes beyond single observed cases:

- Motivation for learning is to deal effectively with ***new*** situations
- Importance of deriving general rules or principles from a limited set of observations or experiences

➔ Central concept: ***GENERALISATION***

Why is Generalisation Important?

Consider a probabilistic reasoning agent:

Could learn the Full Joint Probability Distribution of its world by simply maintaining counts of the relative frequencies of all possible world states that it sees (if the world is discrete/finite …)

➔ If the agent encounters every possible state of the world infinitely often, its probability estimates will converge towards the true joint distribution

BUT: The agent will be dead long before then …
(the full joint distribution contains an exponential number of entries!)

Consequences:

- “Learning by heart” is not sufficient
- Agent must be able to **generalise** (to states it has never seen before)

Fundamental Concept in Learning: Inductive Generalisation
(deriving general rules / laws / principles / hypotheses from a limited set of specific observations)

This part not treated in class – ignore for exam!

Part 9:
Machine Learning

Part 9A:
Learning Concepts and Definitions

(A Logical View of Learning)

Univ.-Prof. Dr. Gerhard Widmer
Department of Computational Perception
Johannes Kepler University Linz

gerhard.widmer@jku.at
http://www.cp.jku.at/people/widmer

This part not treated in class – ignore for exam!

Overview of Part A

Central Notions: Learning and Inductive Generalisation

An Example: Concept Learning via Decision Trees

The ID3 Algorithm

A Practical Application Example

Remember: Fundamental Forms of Logical Inference (Aristotle's Syllogisms)

1. Deduction (modus ponens; logically sound):

known: general rule $p \rightarrow q$ ("whenever p is true, q is also true")

known/observed: p ("p is true")

---

deductive conclusion: q ("q is true")

2. Abduction (not logically sound):

known: general rule $p \rightarrow q$ ("whenever p is true, q is also true")

known/observed: q ("q is true")

---

abductive conclusion: p ("p is true")

e.g., p ≡ "it is Sunday"
q ≡ "all the shops are closed"
r ≡ "it is raining"
s ≡ "it is sunny"
....

This part not treated in class – ignore for exam!

This part not treated in class – ignore for exam!

Remember: Fundamental Forms of Logical Inference (Aristotle's Syllogisms)

3. Induction / Inductive Generalisation (not logically sound *):

observed:	p, q, r	(p, q, and r occurred together)
sets of	$p, q, s, t, \ldots$	($p, q, s, t, \ldots$ occurred together)
facts:	$p, q, s, u, v, \ldots$	
	....	

inductive conclusion:	$p \rightarrow q$	("whenever p is true, q is also true")
or	$q \rightarrow p$	("whenever q is true, p is also true")

e.g., p ≡ "it is Sunday"
q ≡ "all the shops are closed"
r ≡ "it is raining"
s ≡ "it is sunny"
....

*) cf. the *"cum hoc ergo propter hoc" fallacy*

Properties of Inductive Generalisation

- Infers general rules/laws from a finite set of specific observations
- Is not logically justified (unless we have seen *all possible situations*)
- The 'likelihood' of a generalisation to be correct grows with the number of confirming cases, BUT:
- *A single counter-example* can falsify an inductive generalisation

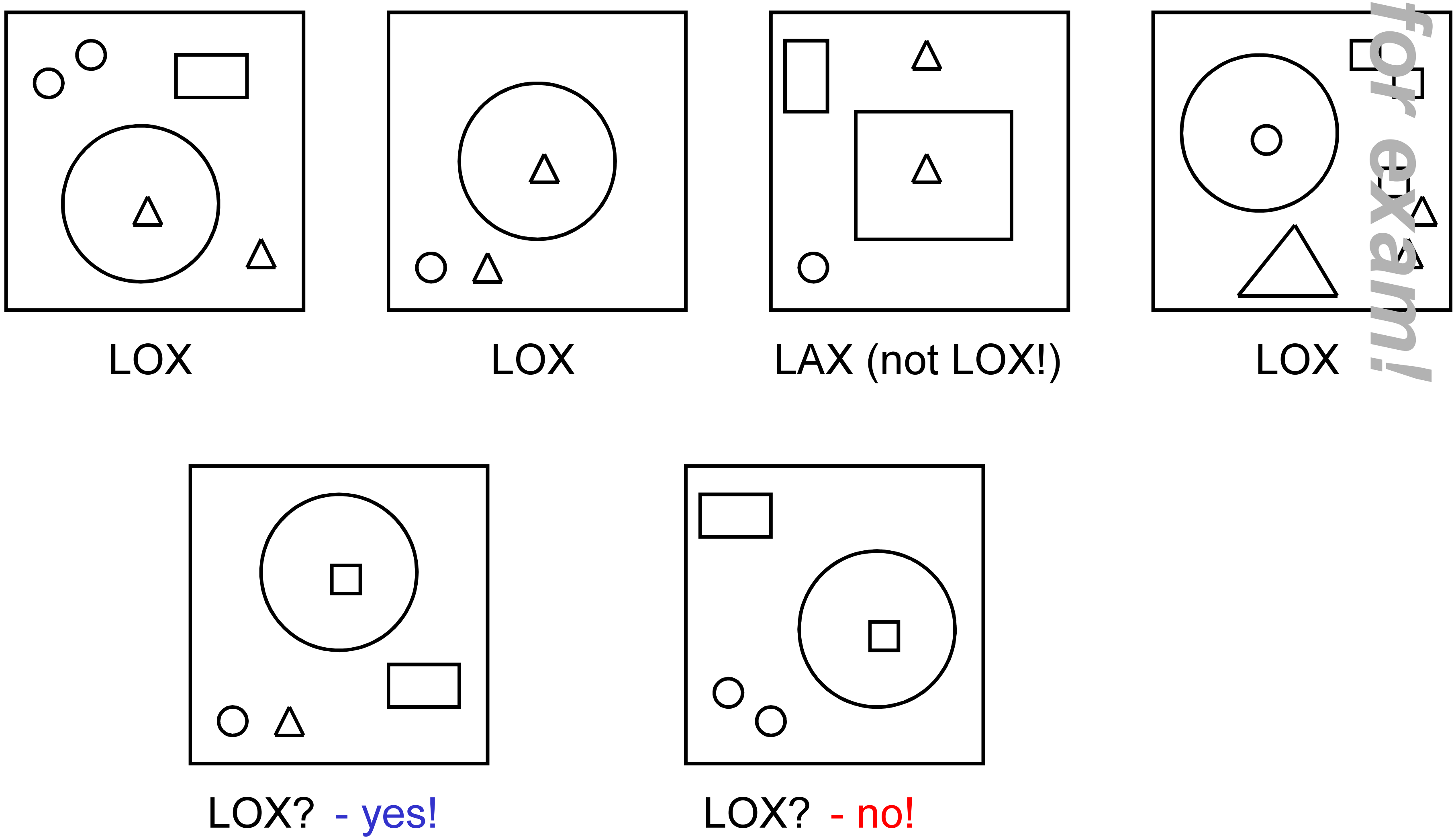
➔ The result of an inductive generalisation is a **hypothesis**

Inductive generalisation is dangerous, but important:

- The only way of producing genuinely new general knowledge (rules)
- All forms of non-trivial learning use inductive inference

This part not treated in class – ignore for exam!

Learning the Definition of a Concept: What Makes a LOX?



This part not treated in class – ignore for exam!

Inductive Generalisation as a Search Problem

Search for a hypothesis (description, definition, ...) that

- includes ('covers', is true for) all positive examples
- excludes all counterexamples

Search for properties that

- are common to all positive examples and
- distinguish them from all negative examples

➔ Properties of a good hypothesis:

- **completeness** (should cover all positive examples)
- **consistency** (should not cover any negative examples)

This part not treated in class – ignore for exam!

Concept Learning: Learning Logical Definitions in the Form of Decision Trees

This part not treated in class – ignore for exam!

Given: Training examples:

A set of positive examples (P) and counter-examples (N) of a given concept C

Learn: A 'hypothesis':

A general definition of the concept C that

- is consistent with all known examples (is true for all positive examples $\in P$ and is false for all negative examples $\in N$)
- will predict a correct classification in all future cases

For now, assume a simple "propositional" representation:

- An example is described by a list of n attributes (properties), each of which can take one of a finite set of possible (discrete) values

➔ An example is a list of attribute values + given classification

A Simple Discrete Classification Problem

This part not treated in class – ignore for exam!

Example observations:

Attributes/Features — **Class**

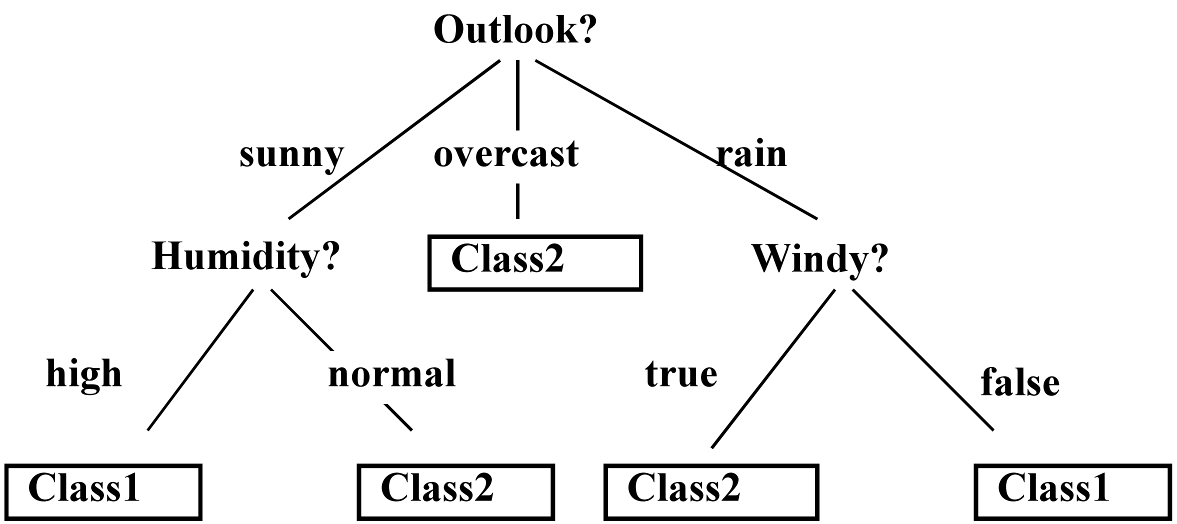
Examples

Day	Outlook	Temp.	Humidity	Windy	CLASS
day1	sunny	hot	high	false	Don't Play
day2	sunny	hot	high	true	Don't Play
day3	overcast	hot	high	false	Play
day4	rain	mild	high	false	Play
day5	rain	cool	normal	false	Play
day6	rain	cool	normal	true	Don't Play
day7	overcast	cool	normal	true	Play
day8	sunny	mild	high	false	Don't Play
day9	sunny	cool	normal	false	Play
day10	rain	mild	normal	false	Play
day11	sunny	mild	normal	true	Play
day12	overcast	mild	high	true	Play
day13	overcast	hot	normal	false	Play
day14	rain	mild	high	true	Don't Play

➔ Can you learn a general rule from this that can predict when this person will play tennis?

What is a Decision Tree?

A hierarchical decision structure ...



... equivalent to a set of logical classification rules (implications):

IF Outlook=sunny AND Humidity=high THEN Prediction=Class1
IF Outlook=sunny AND Humidity=normal THEN Prediction=Class2
IF Outlook=overcast THEN Prediction=Class2
IF Outlook=rain AND Windy=true THEN Prediction=Class2
IF Outlook=rain AND Windy=false THEN Prediction=Class1

This part not treated in class – ignore for exam!

What Conditions Should a Decision Tree Satisfy?

- Should be **consistent** with training data
  (should predict correct class for each known training example)

- Should be a **generalisation** (i.e., more general than the sum of the given examples) ➔ make predictions on new, unseen cases!

- Problem: In general, one can construct a **huge number** of decision trees from a given set of examples that are all consistent with the given examples (and are generalisations of various degrees) !

  … try it on our 14 “Play/Don’tPlay” examples …

- The tree we seek should be the **“correct”** one, i.e., the one that
  - represents the true explanation of the observed events
  - make correct predictions on new cases in the future!

This part not treated in class – ignore for exam!

What Conditions Should a Decision Tree Satisfy?

This part not treated in class – ignore for exam!

- Should be **consistent** with training data
  (should predict correct class for each known training example)

- Should be a **generalisation** (i.e., more general than the sum of the given examples) ➔ make predictions on new, unseen cases!

- ~~Should be **"correct"**, i.e., make correct predictions on new cases in the future!~~
  [untestable]

- Should be as **simple** as possible
  [Motivation: see next slide]

  ➔ Search for simplest tree consistent with given training examples
  ➔ Combinatorial complexity — exhaustive search impossible!
  ➔ Heuristic (greedy) search algorithm **ID3**

"Occam's Razor" Principle

This part not treated in class – ignore for exam!

"Non sunt multiplicanda entia praeter necessitatem."
(Entities should not be multiplied beyond necessity.)
William of Ockham (1290? - 1349?)

Source:
B. Natarajan (1991): *Machine Learning: A Theoretical Approach* (p.51).
San Francisco, CA: Morgan Kaufmann.

According to Bertrand Russell, the actual phrase used by William of Ockham was:

"It is vain to do with more what can be done with fewer."

M. Li & P. Vitányi (1993): *An Introduction to Kolmogorov Complexity and its Applications* (p.276-277). Berlin: Springer.

General interpretation of Ockham's principle in science:

"Among the theories consistent with observed phenomena, one should prefer the simplest theory."

Not to be confused with Newton's (stronger) statement:

"Natura enim simplex est, et rerum causis superfluis non luxuriat."
(Nature is simple, and does not afford the pomp of superfluous causes.)

I. Newton, Preface to *"Principia"*
(cited after Li & Vitanyi, 1993)

The ID3 Algorithm for Learning Decision Trees

- **Recursive** algorithm:
  Builds a decision tree step by step;
  starts with empty tree
- **Heuristic** algorithm:
  Aims at constructing a simple tree, but cannot guarantee that it will find the simplest tree (that would require constructing all possible trees and choosing the simplest one!)
- **Greedy** algorithm:
  At each step, tries to make decision (what attribute to choose next) that maximises a local heuristic optimality criterion (information gain);
  blind to attributes that are relevant only in combination

This part not treated in class – ignore for exam!

This part not treated in class – ignore for exam!

The ID3 Algorithm: Top-down Induction of Decision Trees

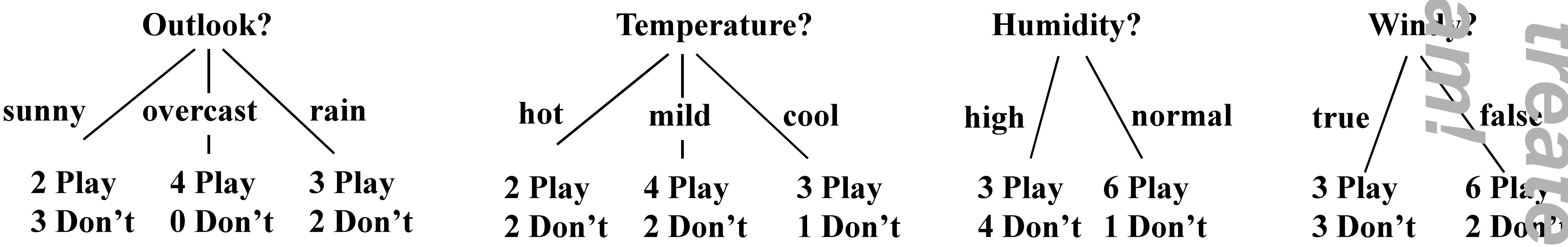
- Stepwise, recursive construction of a decision tree
- Start with root node and all given training examples

Main loop:

1. If all examples in current node belong to the same class C
   ➔ Make current node a leaf (with class label C) and EXIT loop
2. Select attribute A that is "best" for current node, and assign A as decision attribute to current node
3. Create a branch + successor node for each possible value of A
4. Split training examples associated with current node into subsets according to their value of A; assign each subset to its respective branch/subnode
5. For each subnode: recursively call ID3

What is the “Best” Attribute?

Intuition:
“Best” attribute is the one that best discriminates between classes and thus is most likely to produce subsets of examples that will lead to a simple tree



Formalisation of this idea in a numeric quality measure that judges the discrimination ability of attributes:

Information Gain

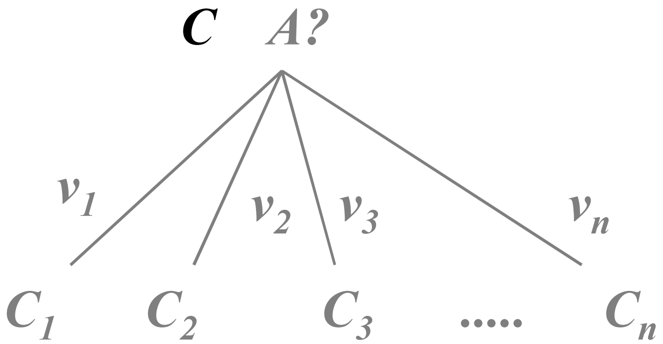
This part not treated in class – ignore for exam!

This part not treated in class – ignore for exam!

Entropy as a Measure of Class Separation

Notation (assumption: 2 classes):

- A ... some (discrete) attribute with possible values $v_1, ..., v_n$
- C ... set of training examples associated with current node
- N ... number of examples in C ($N = |C|$)
- p, n ... number of positive / negative examples in C: $p+n = N$
- p_i, n_i ... number of positive / negative examples in subnode C_i

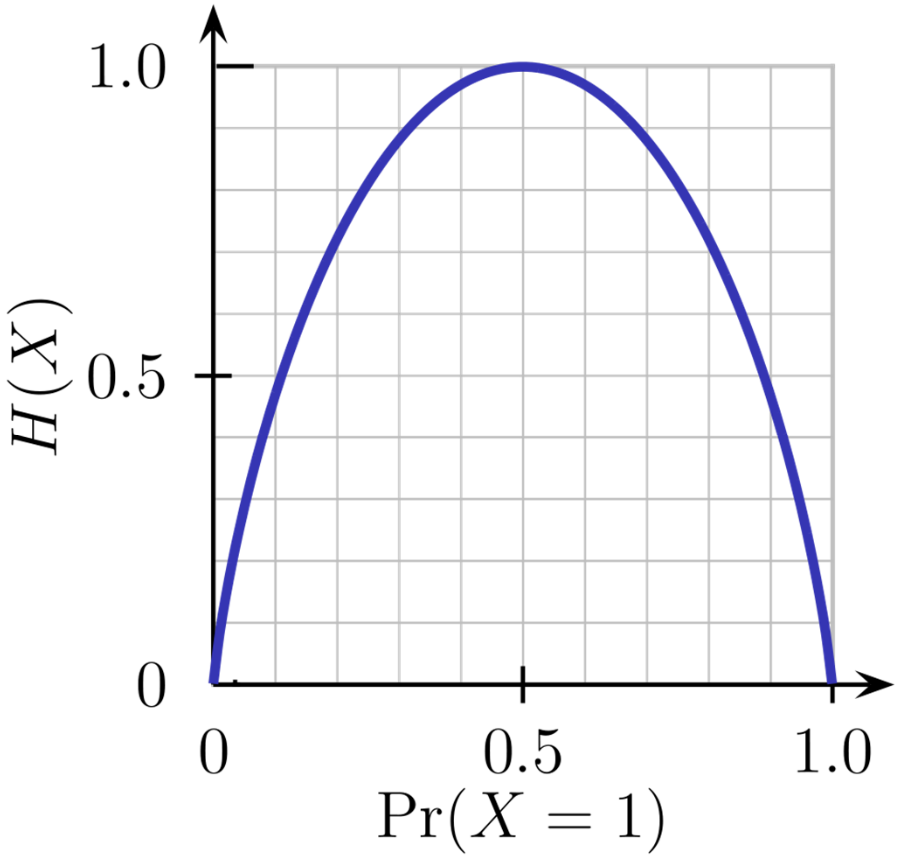


Definition 1:

$$Entropy(C) = - p/(p+n) \log_2 p/(p+n) - n/(p+n) \log_2 n/(p+n)$$

➔ Entropy is a measure of how uniform vs. skewed the distribution of class labels is in a set C of labels (labelled objects)

Entropy



Binary Entropy Function depending on p

Entropy of a Bernoulli trial as a function of success probability, often called the ***binary entropy function****. The entropy is maximized at 1 bit per trial when the two possible outcomes are equally probable, as in an unbiased coin toss.*

Entropy

$$H(X) = \mathbb{E}_X[I(x)] = -\sum_{x \in \mathbb{X}} p(x) \log p(x).$$

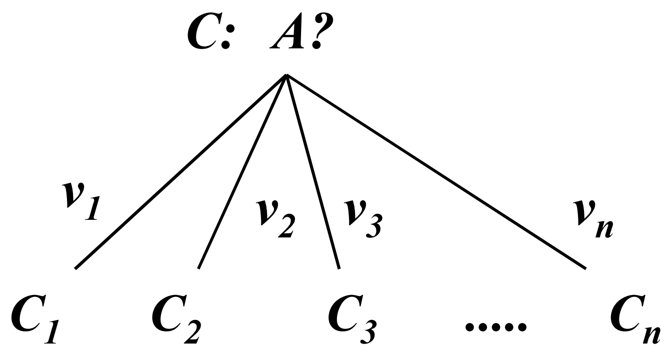
Binary Entropy

$$H_\mathrm{b}(p) = -p \log_2 p - (1-p) \log_2 (1-p).$$

➔ Entropy is a measure of the "impurity" of set C with respect to class labels
➔ Want to **minimise** this (want leaves that contain only instances of one class)

This part not treated in class – ignore for exam!

Information Gain



Definition 2: $\color{red}{InfoGain(C,A)} = Entropy(C) - \sum |C_i|/|C| * Entropy(C_i)$

➔ **InfoGain** is **expected reduction in entropy** if data are split according to A

➔ **ID3 selects the attribute with the largest InfoGain**

Note:

Entropy(*C*) is independent of *A*

➔ Maximising *InfoGain* is equivalent to minimising the second term (weighted sum of entropies of subsets)

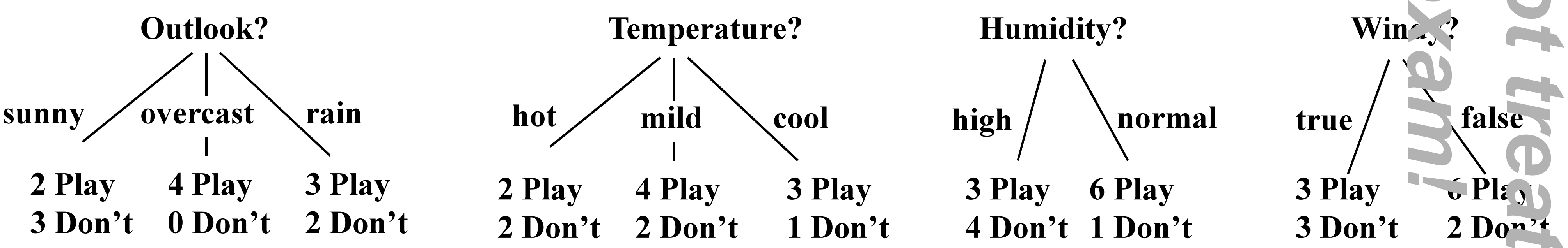
This part not treated in class – ignore for exam!

An Example

Step 1: Selecting the root attribute:

Weight

$$\mathit{InfoGain}(C,A) = \mathit{Entropy}(C) - \sum |C_i|/|C| * \mathit{Entropy}(C_i)$$



Class distribution at the root (i.e., in the full training set): **9 Play, 5 Don't Play**

➔ $\mathit{Entropy}(C) = - 9/14 * \mathit{log}_2(9/14) - 5/14 * \mathit{log}_2(5/14) = 0.940$ [bits]

$\mathit{Gain}(C,\mathit{Outlook}) = .940 - 5/14*.971 - 4/14*0.00 - 5/14*.971 = 0.246$ [bits] **max.**

$\mathit{Gain}(C,\mathit{Temperature}) = .940 - 4/14*1.00 - 6/14*.918 - 4/14*.811 = 0.029$ [bits]

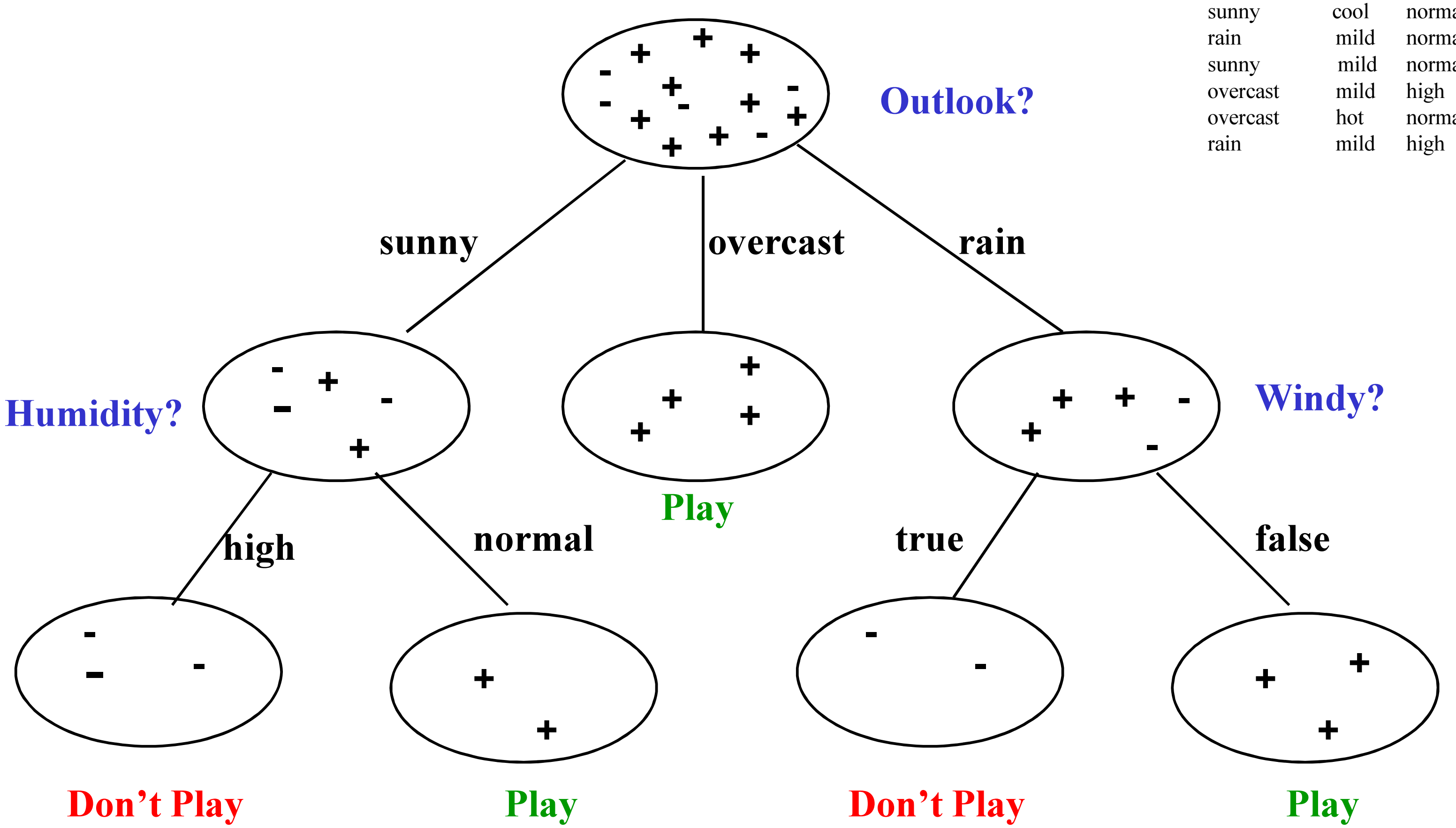
$\mathit{Gain}(C,\mathit{Humidity}) = .940 - 7/14*.985 - 7/14*.592 = 0.151$ [bits]

$\mathit{Gain}(C,\mathit{Windy}) = .940 - 6/14*1.00 - 8/14*.811 = 0.048$ [bits]

This part not treated in class – ignore for exam!

The Final Tree

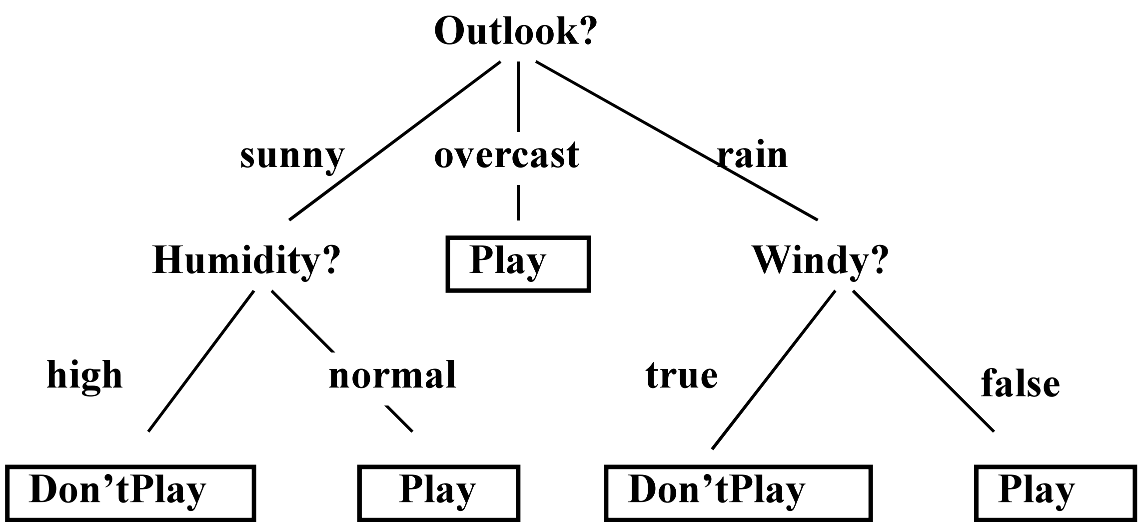
Outlook	Temp.	Humidity	Windy	CLASS
Sunny	hot	high	false	Don't Play
sunny	hot	high	true	Don't Play
overcast	hot	high	false	Play
rain	mild	high	false	Play
rain	cool	normal	false	Play
rain	cool	normal	true	Don't Play
overcast	cool	normal	true	Play
sunny	mild	high	false	Don't Play
sunny	cool	normal	false	Play
rain	mild	normal	false	Play
sunny	mild	normal	true	Play
overcast	mild	high	true	Play
overcast	hot	normal	false	Play
rain	mild	high	true	Don't Play



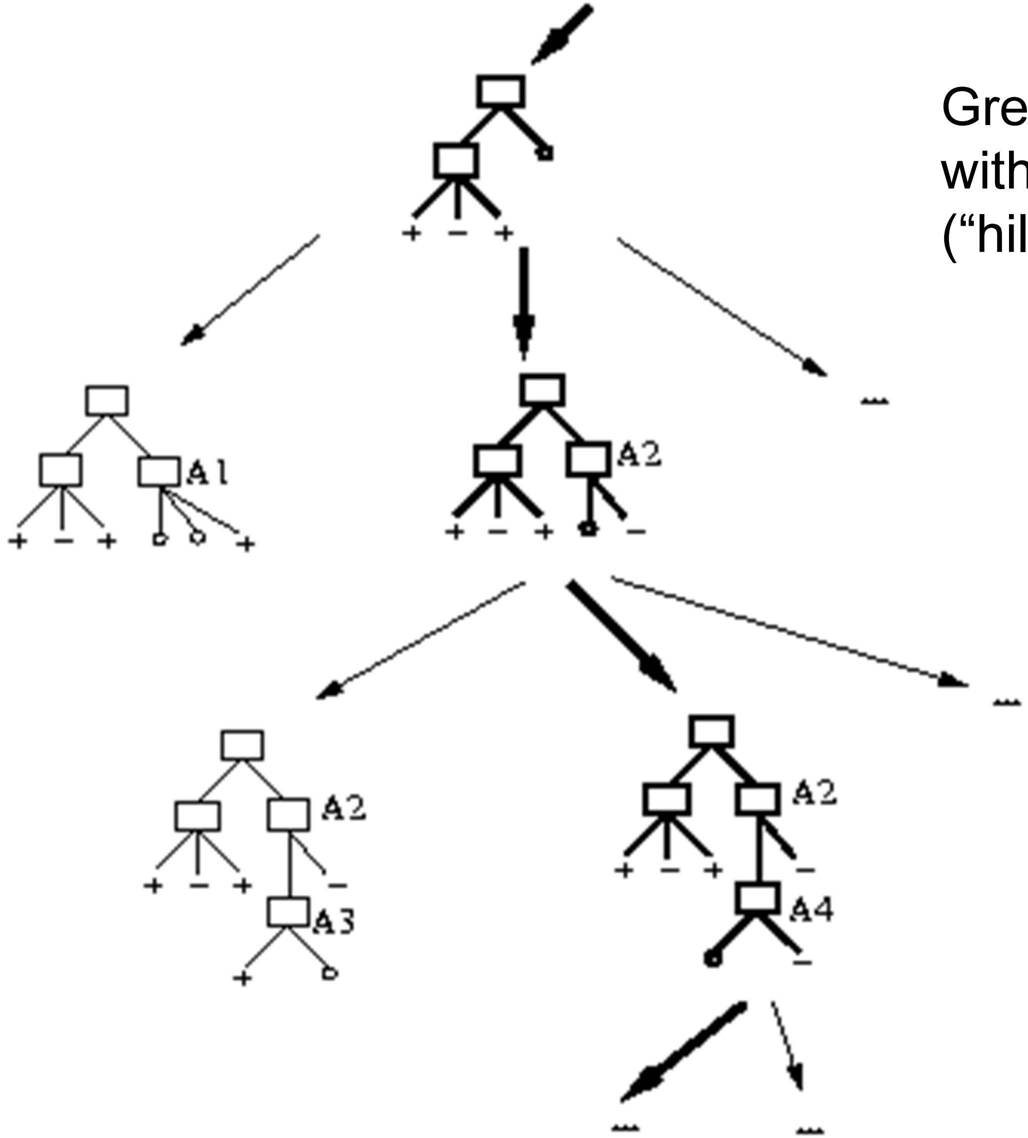
This part not treated in class – ignore for exam!

This part not treated in class – ignore for exam!

Outlook	Temp.	Humidity	Windy	CLASS
sunny	hot	high	false	Don't Play
sunny	hot	high	true	Don't Play
overcast	hot	high	false	Play
rain	mild	high	false	Play
rain	cool	normal	false	Play
rain	cool	normal	true	Don't Play
overcast	cool	normal	true	Play
sunny	mild	high	false	Don't Play
sunny	cool	normal	false	Play
rain	mild	normal	false	Play
sunny	mild	normal	true	Play
overcast	mild	high	true	Play
overcast	hot	normal	false	Play
rain	mild	high	true	Don't Play



Hypothesis Space Search by ID3



Greedy local search without backtracking ("hill-climbing search")

This part not treated in class – ignore for exam!

Numeric Attributes

Outlook	Temp.	Humidity	Windy	CLASS
sunny	85	high	false	Don't Play
sunny	80	high	true	Don't Play
overcast	83	high	false	Play
rain	70	high	false	Play
rain	68	normal	false	Play
rain	65	normal	true	Don't Play
overcast	64	normal	true	Play
sunny	72	high	false	Don't Play
sunny	69	normal	false	Play
rain	75	normal	false	Play
sunny	75	normal	true	Play
overcast	72	high	true	Play
overcast	81	normal	false	Play
rain	71	high	true	Don't Play

How to split on Temperature? And how to compute InfoGain(Temperature)?
(Do not want to create a branch for every possible numeric value!)

This part not treated in class – ignore for exam!

Numeric Attributes

Idea: create binary tests ($A \leq x$, $A > x$) for numeric attributes

Algorithm for determining best split for attribute A:

- Sort examples according to their value for A
- Tentatively use each mid-point x_i between two values as a possible split point
- Compute $\boldsymbol{InfoGain(A \leq x_i)}$ that would be obtained if this split point were chosen
- Select x_i with maximum info gain; this test competes with all the other attributes (discrete or numeric) for the best attribute overall.

Temperature:	**64**	**65**	**68**	**69**	**70**	**71**	**72**	**75**	**80**	**81**	**83**	**85**
Class: Play	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**2**	**0**	**1**	**1**	**0**
Don't Play	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**

Temp ≤ 70.5?

true → **4 P, 1 D**

false → **5 P, 4 D**

This part not treated in class – ignore for exam!

A Practical Application in the Steel Industry

Given:

Example data (measurements) from a steel production plant (properties of input materials, process parameters, quality of resulting product)

Learn:

Rules that can predict under what circumstances the product will be faulty

This part not treated in class – ignore for exam!

0.04,3.0,0.17,0.018,0.021919,1.409912,0.001,0.001,0.007998,0.006,0.21,0.002,0.002,0.002,3.0,145.55078,0.41,5.890869,1305.375,1306.1
875,2.0,469.0,469.0,13.172852,98.816406,0.5,0.0,86.167969,19.201843,1.210938,1.208984,0.19,48.0,31.0,0.17,0.19,0.69,1.299805,41[illegible]7
5,2.156982,4.812988,2.656006,0.79,1.387939,0.5989,0.83,1.336914,0.5,53.628906,21959.0,15385.0,32957.0,59.0,4.0,361.0,4,4,1,15,1.76,
2,8,50,2,2,2,150,2,1,2,1,1,2,2,2,2.724853,1.92,0.00125,0.00531931333,0.19,1.299805,2, NEGATIVE

0.044949,3.0,0.17,0.025,0.023,1.429932,0.004999,0.011,0.011919,0.006,0.18,0.002,0.002,0.001,126.18359,131.16406,4.942871,4.97998,13
05.75,1306.5625,2.0,18.0,18.0,97.710938,97.890625,0.5,0.0,4.036865,0.5,0.008999,1.198975,1.196777,37.0,29.0,0.19,-
0.19,1.799805,1.399902,4493.125,3.948975,4.154785,0.2,1.22583,1.280762,0.054949,1.058838,1.10083,0.041919,46.496094,25358.0,17087.0
,34659.0,18.0,2.0,439.0,4,4,1,61,2,93,2,8,44,2,1,2,150,2,2,2,2,2,2,2,2,2.381592,1.663113,0.0085005001,0.00550050,0.19,1.799805,2
POSITIVE

0.044,1.0,0.1,0.03,0.046,0.45,0.004,0.002,0.007,0.009998,0.014949,0.004999,0.001,0.001,75.980469,85.207031,5.72583,5.743896,12[illegible]0,
1303.0,1.458333,668.0,668.0,69.824219,105.38281,0.44,0.005997,5.906982,2.836385,0.12,1.293945,1.183838,51.0,37.0,0.22,0.5,11.199951
,1.799805,4182.25,3.391846,3.87085,0.47,1.024902,1.11499,0.09,0.94,1.056885,0.1,49.101563,26153.0,19479.0,19479.0,623.0,[illegible].0,1014.0,
4,5,2,61,2,80,2,8,51,3,3,1,130,2,2,2,2,2,2,2,2,2.171875,2.2438,0.00175,0.0118185001,0.5,11.199951,2, POSITIVE

0.037979,1.0,0.09595,0.02,0.09595,0.57,0.001,0.001,0.011919,0.008999,0.016969,0.004999,0.001,0.001,131.875,137.07422,5.134766,6.[illegible]74
902,1276.1875,1315.5,1.0,140.0,140.0,70.414063,94.699219,0.5,0.003998,4.042969,2.01062,0.3,1.253906,0.98,30.0,34.734375,0.122,-
0.19,1.699951,3.099854,4487.1667,3.330811,3.470947,0.14,0.98,1.076904,0.090909,0.86,0.93,0.067979,51.285156,20956.0,14328.0,31900.0
,129.0,1.0,274.0,4,4,1,61,2,84,2,8,45,2,2,1,130,2,2,2,2,2,2,2,2,2.010237,1.13937,0.001,0.0201011666,0.19,3.099854,0, POSITIVE

0.046,1.0,0.2,0.028,0.021,0.62222,0.002,0.003,0.006,0.004999,0.018989,0.002,0.001,0.002,115.04297,116.27344,4.846924,4.[illegible]4805,1525.
0,1525.0,2.0,726.0,726.0,105.09766,105.87891,0.49,0.0,2.927979,1.736542,0.021,1.299805,1.291992,44.0,34.0,0.22,0.5,11.199951,1.7998
05,4194.6667,5.336914,5.517822,0.18,1.250977,1.324951,0.073939,1.195801,1.295898,0.1,38.660156,26211.0,19537.0,19537.0,681.0,2.0,10
72.0,4,5,2,61,2,80,2,8,51,3,3,2,750,2,1,2,2,2,2,2,2,2.6208489998,2.024,0.00275075,0.0051661667,0.5,11.199951,2, NEGATIVE

0.049,1.0,0.079,0.037979,0.066969,0.47,0.003,0.002,0.014,0.008999,0.19,0.007,0.003,0.003,92.3,89.09,6.233887,6.315918,1285.0,1285.0
,2.0,342.0,2055.0,102.85156,104.42969,0.44,0.0,8.595947001,5.599264,0.016,1.282959,1.278809,40.0,28.0,5988989,0.59100,0.0,0.0,4504.
1667,3.709961,3.75,0.04,0.999,0.999,0.099,1.014,0.977,0.1,46.488281,1524.0,16256.0,20786.0,146.0,2.0,2.0,3,3,1,61,2,84,2,8,45,2,[illegible],2
,160,2,2,2,2,2,2,2,2,1.976,1.96,0.00225025,0.0169698334,0.59100,0.0,2, NEGATIVE

0.051,0.0,0.16,0.028,0.054,0.8,0.002,0.002,0.008999,0.011919,0.014949,0.004,0.002,0.003,122.63672,127.74609,4.557861,4.608887,12[illegible]0.
0,1285.0,1.791667,139.0,139.0,81.035156,98.09375,0.49,0.002014,3.664795,1.784215,0.2,1.198975,0.,34.0,35.042969,0.22,0.19,1.199951,
2.099854,5557.9167,2.907959,3.12085,0.21,1.091797,1.169922,0.078181,0.93,1.031982,0.094949,44.484375,18946.0,10394.0,27966.0,139.0,
2.0,459.0,4,4,1,61,2,84,2,8,42,2,2,1,530,2,2,2,2,2,2,2,2,2.201904,1.734,0.00225025,0.012323,0.22,2.099854,2, POSITIVE

0.047,1.0,0.17,0.018989,0.03,0.91,0.002,0.002,0.007998,0.011919,0.016,0.004,0.001,0.003,116.21875,118.28125,4.960938,5.001953,1116.
5,1116.875,2.0,402.0,402.0,97.375,97.472656,0.5,0.001007,4.838867,1.007039,0.004999,1.193848,1.192871,41.0,31.210938,0.29,0.19,0.59
100,1.899902,5535.5833,2.579834,2.636963,0.057171,1.147949,1.171875,0.023939,0.96,0.,0.028,48.160156,17254.0,12391.0,29963.0,398.0,
4.0,181.0,4,4,1,61,2,84,2,8,48,2,2,2,530,2,2,2,2,2,2,2,2,2.171874,1.927,0.00225025,0.008333,0.29,1.899902,2, NEGATIVE

0.044949,1.0,0.17,0.018,0.026969,0.92,0.007,0.003,0.004999,0.009998,0.014949,0.003,0.001,0.001,42.484375,53.488281,6.901855,6.[illegible]86
3,1310.0,1310.0,2.0,530.0,100.0,58.175781,97.054688,0.5033979,0.0,9.265869,3.089661,0.5033979,1.192871,0.72222,31.0,27.0,0.1,0.5,1.
0,1.099854,4468.7917,2.876953,3.554932,0.67,0.86,1.061768,0.19,0.78,0.96,0.17,36.503906,24466.0,16710.0,16710.0,94.0,8.0,682.0,4,5,
2,61,2,80,2,8,42,2,3,2,530,2,1,2,2,2,2,2,2,2.028434,1.39341902,0.00250050,0.0069949933,0.5,1.099854,2, POSITIVE

0.044949,1.0,0.1,0.013,0.018989,0.54,0.001,0.001,0.004999,0.004999,0.018989,0.002,0.001,0.002,116.85938,119.35547,6.022949,6.068848
,1589.875,1590.25,2.0,368.0,368.0,96.03125,96.230469,0.45,0.0,4.268799,2.114001,0.008999,1.175781,1.173828,40.0,32.84375,0.122,0.19
,0.69,2.0,4200.8696,4.402832,4.552979,0.15,1.001953,1.025879,0.023939,0.91,0.94,0.03303,52.601563,21189.0,14561.0,32133.0,358.0,2.0
,173.0,4,4,1,61,2,80,2,8,45,2,2,2,130,2,2,2,2,2,2,2,2,1.970371,1.79796002,0.00125,0.0048318,0.19,2.0,2, NEGATIVE

This part not treated in class - Ignore for exam!

A Learned Decision Tree

```
V76 = 1: 0 (21.0/1.0)
V76 = 2
|   V5 <= 0.090909
|   |   V63 = 85: 1 (1.0)
|   |   V63 = 95: 0 (0.0)
|   |   V63 = 93: 0 (4.0)
|   |   V63 = 78: 0 (0.0)
|   |   V63 = 90: 0 (1.0)
|   |   V63 = 76
|   |   |   V31 <= 2.028615: 1 (4.0)
|   |   |   V31 > 2.028615: 0 (9.0)
|   |   V63 = 103: 0 (0.0)
|   |   V63 = 91
|   |   |   V7 <= 0.018989: 1 (3.0)
|   |   |   V7 > 0.018989: 0 (18.0)
|   |   V63 = 84
|   |   |   MC = 1
|   |   |   |   V30 <= 4.679932: 0 (5.0)
|   |   |   |   V30 > 4.679932: 1 (3.0)
|   |   |   MC = 0: 0 (5.0)
|   |   |   MC = 2: 0 (49.0/1.0)
|   |   V63 = 94
|   |   |   V10 <= 0.002
|   |   |   |   V21 <= 1307.8125: 0 (1.0)
|   |   |   |   V21 > 1307.8125: 1 (6.0)
|   |   |   V10 > 0.002: 0 (3.0)
|   |   V63 = 92
|   |   |   V28 <= 0.444444: 1 (2.0)
|   |   |   V28 > 0.444444: 0 (3.0)
|   |   V63 = 80
|   |   |   V15 <= 0.002
|   |   |   |   V54 <= 12774: 1 (4.0)
|   |   |   |   V54 > 12774
|   |   |   |   |   V11 <= 0.007
|   |   |   |   |   |   V15 <= 0.001
|   |   |   |   |   |   |   V49 <= 1.077881
|   |   |   |   |   |   |   |   V5 <= 0.078989: 1 (1.0)
|   |   |   |   |   |   |   |   V5 > 0.078989: 0 (8.0)
|   |   |   |   |   |   |   V49 > 1.077881: 1 (4.0/1.0)
|   |   |   |   |   |   V15 > 0.001: 1 (1.0)
|   |   |   |   |   V11 > 0.007: 0 (19.0/1.0)
|   |   |   V15 > 0.002: 1 (2.0)
|   V5 > 0.090909
|   |   V12 <= 0.009999
|   |   |   V62 = 1: 0 (6.0)
|   |   |   V62 = 2
|   |   |   |   V40 <= 0
|   |   |   |   |   V38 <= -0.19999: 1 (4.0)
|   |   |   |   |   V38 > -0.19999
|   |   |   |   |   |   V10 <= 0.002
|   |   |   |   |   |   |   V35 <= 48
|   |   |   |   |   |   |   |   V37 <= 0.155555
|   |   |   |   |   |   |   |   |   V67 = 44
|   |   |   |   |   |   |   |   |   |   V3 <= 0.016: 0 (1.0)
|   |   |   |   |   |   |   |   |   |   V3 > 0.016: 1 (3.0)
|   |   |   |   |   |   |   |   |   V67 = 52
|   |   |   |   |   |   |   |   |   |   V5 <= 0.1: 0 (1.0)
|   |   |   |   |   |   |   |   |   |   V5 > 0.1: 1 (2.0)
|   |   |   |   |   |   |   |   |   V67 = 48: 1 (0.0)
|   |   |   |   |   |   |   |   |   V67 = 43: 0 (2.0)
|   |   |   |   |   |   |   |   |   V67 = 45
|   |   |   |   |   |   |   |   |   |   V56 <= 3
|   |   |   |   |   |   |   |   |   |   |   V23 <= 1.583333: 1 (1.0)
|   |   |   |   |   |   |   |   |   |   |   V23 > 1.583333: 0 (3.0)
|   |   |   |   |   |   |   |   |   |   V56 > 3: 1 (2.0)
|   |   |   |   |   |   |   |   |   V67 = 50: 0 (1.0)
|   |   |   |   |   |   |   |   |   V67 = 49: 1 (0.0)
|   |   |   |   |   |   |   |   |   V67 = 51: 1 (0.0)
|   |   |   |   |   |   |   |   |   V67 = 42: 1 (0.0)
|   |   |   |   |   |   |   |   |   V67 = 46: 1 (2.0)
|   |   |   |   |   |   |   |   |   V67 = 41
|   |   |   |   |   |   |   |   |   |   V36 <= 20: 0 (1.0)
|   |   |   |   |   |   |   |   |   |   V36 > 20: 1 (2.0)
|   |   |   |   |   |   |   |   V37 > 0.155555: 1 (15.0)
|   |   |   |   |   |   |   V35 > 48
|   |   |   |   |   |   |   |   V29 <= 0.01303: 0 (7.0)
|   |   |   |   |   |   |   |   V29 > 0.01303: 1 (1.0)
|   |   |   |   |   |   V10 > 0.002: 0 (3.0)
|   |   |   |   V40 > 0
|   |   |   |   |   V16 <= 0.001
|   |   |   |   |   |   V35 <= 30: 1 (6.0)
|   |   |   |   |   |   V35 > 30
|   |   |   |   |   |   |   AL-N <= 2.162
|   |   |   |   |   |   |   |   V77 = 1: 1 (1.0)
|   |   |   |   |   |   |   |   V77 = 2: 0 (27.0/5.0)
|   |   |   |   |   |   |   AL-N > 2.162: 1 (6.0)
|   |   |   |   |   V16 > 0.001
|   |   |   |   |   |   V71 = 110: 0 (0.0)
|   |   |   |   |   |   V71 = 160: 0 (0.0)
|   |   |   |   |   |   V71 = 150: 0 (0.0)
|   |   |   |   |   |   V71 = 350
|   |   |   |   |   |   |   V15 <= 0.002: 0 (5.0)
|   |   |   |   |   |   |   V15 > 0.002
|   |   |   |   |   |   |   |   V46 <= 1.082764: 1 (4.0)
|   |   |   |   |   |   |   |   V46 > 1.082764
|   |   |   |   |   |   |   |   |   V23 <= 1.458333: 1 (1.0)
|   |   |   |   |   |   |   |   |   V23 > 1.458333
|   |   |   |   |   |   |   |   |   |   V40 <= 1: 1 (1.0)
|   |   |   |   |   |   |   |   |   |   V40 > 1
|   |   |   |   |   |   |   |   |   |   |   V43 <= 3.131836: 1 (1.0)
|   |   |   |   |   |   |   |   |   |   |   V43 > 3.131836: 0 (10.0)
|   |   |   |   |   |   V71 = 750: 0 (2.0)
|   |   |   |   |   |   V71 = 530: 0 (9.0)
|   |   |   |   |   |   V71 = 550: 0 (0.0)
|   |   |   |   |   |   V71 = 130
|   |   |   |   |   |   |   V75 = 1: 1 (1.0)
|   |   |   |   |   |   |   V75 = 2
|   |   |   |   |   |   |   |   V24 <= 145
|   |   |   |   |   |   |   |   |   V14 <= 0.004: 0 (1.0)
|   |   |   |   |   |   |   |   |   V14 > 0.004: 1 (2.0)
|   |   |   |   |   |   |   |   V24 > 145: 0 (29.0)
|   |   V12 > 0.009999
|   |   |   V79 = 1: 1 (6.0)
|   |   |   V79 = 2
|   |   |   |   V68 = 2
|   |   |   |   |   V72 = 1: 0 (1.0)
|   |   |   |   |   V72 = 2
|   |   |   |   |   |   V63 = 85: 1 (0.0)
|   |   |   |   |   |   V63 = 95: 1 (0.0)
|   |   |   |   |   |   V63 = 93: 0 (1.0)
|   |   |   |   |   |   V63 = 78: 1 (0.0)
|   |   |   |   |   |   V63 = 90: 1 (0.0)
|   |   |   |   |   |   V63 = 76: 1 (3.0)
|   |   |   |   |   |   V63 = 103: 1 (0.0)
|   |   |   |   |   |   V63 = 91
|   |   |   |   |   |   |   V40 <= 1.099854
|   |   |   |   |   |   |   |   V69 = 1: 0 (0.0)
|   |   |   |   |   |   |   |   V69 = 3
|   |   |   |   |   |   |   |   |   V30 <= 6.246826: 0 (2.0)
|   |   |   |   |   |   |   |   |   V30 > 6.246826: 1 (2.0)
|   |   |   |   |   |   |   |   V69 = 2: 0 (3.0)
|   |   |   |   |   |   |   V40 > 1.099854: 1 (3.0)
|   |   |   |   |   |   V63 = 84
|   |   |   |   |   |   |   V73 = 2
|   |   |   |   |   |   |   |   V58 = 4
|   |   |   |   |   |   |   |   |   V55 <= 99: 1 (10.0)
|   |   |   |   |   |   |   |   |   V55 > 99
|   |   |   |   |   |   |   |   |   |   AL-N <= 1.708062: 0 (6.0)
|   |   |   |   |   |   |   |   |   |   AL-N > 1.708062
|   |   |   |   |   |   |   |   |   |   |   V32 <= 0.004999: 0 (2.0)
|   |   |   |   |   |   |   |   |   |   |   V32 > 0.004999
|   |   |   |   |   |   |   |   |   |   |   |   V4 <= 1: 1 (6.0)
|   |   |   |   |   |   |   |   |   |   |   |   V4 > 1: 0 (1.0)
|   |   |   |   |   |   |   |   V58 = 3: 1 (10.0)
|   |   |   |   |   |   |   V73 = 1
|   |   |   |   |   |   |   |   V11 <= 0.006: 1 (2.0)
|   |   |   |   |   |   |   |   V11 > 0.006: 0 (7.0)
|   |   |   |   |   |   V63 = 94: 1 (1.0)
|   |   |   |   |   |   V63 = 92: 1 (2.0)
|   |   |   |   |   |   V63 = 80
|   |   |   |   |   |   |   V35 <= 31: 0 (4.0)
|   |   |   |   |   |   |   V35 > 31
|   |   |   |   |   |   |   |   V32 <= 0.013
|   |   |   |   |   |   |   |   |   V4 <= 0: 1 (3.0)
|   |   |   |   |   |   |   |   |   V4 > 0
|   |   |   |   |   |   |   |   |   |   V10 <= 0.001
|   |   |   |   |   |   |   |   |   |   |   V48 <= 0.855555: 1 (2.0)
|   |   |   |   |   |   |   |   |   |   |   V48 > 0.855555: 0 (3.0)
|   |   |   |   |   |   |   |   |   |   V10 > 0.001: 0 (2.0)
|   |   |   |   |   |   |   |   V32 > 0.013: 1 (12.0)
|   |   |   |   V68 = 3
|   |   |   |   |   V9 <= 0.009999: 1 (9.0)
|   |   |   |   |   V9 > 0.009999: 0 (1.0)
```

= Evaluation on test set (200 cases) =

Correctly Classified: 116 58 %
Incorrectly Classified: 84 42 %

=== Confusion Matrix ===

```
 a  b  ← classified as
82 38 |  a = 0
46 34 |  b = 1
```

This part not treated in class – ignore for exam!

A Simpler Tree …

```
V76 = 1: positive (21.0/1.0)
V76 = 2
|   V5 <= 0.090909: positive (156.0/32.0)
|   V5 > 0.090909
|   |   V12 <= 0.009999
|   |   |   V40 <= 0
|   |   |   |   V35 <= 48: negative (42.0/11.0)
|   |   |   |   V35 > 48: positive (9.0/1.0)
|   |   |   V40 > 0
|   |   |   |   V16 <= 0.001
|   |   |   |   |   V35 <= 30: negative (7.0/1.0)
|   |   |   |   |   V35 > 30
|   |   |   |   |   |   AL-N <= 2.162
|   |   |   |   |   |   |   V52 <= 21455
|   |   |   |   |   |   |   |   V25 <= 1129: positive (6.0)
|   |   |   |   |   |   |   |   V25 > 1129: negative (5.0)
|   |   |   |   |   |   |   V52 > 21455: positive (19.0/1.0)
|   |   |   |   |   |   AL-N > 2.162: negative (6.0)
|   |   |   |   V16 > 0.001: positive (69.0/10.0)
|   |   V12 > 0.009999
|   |   |   V70 = 1
|   |   |   |   V24 <= 628: negative (52.0/10.0)
|   |   |   |   V24 > 628: positive (6.0/1.0)
|   |   |   V70 = 2
|   |   |   |   V5 <= 0.094949
|   |   |   |   |   V35 <= 48: positive (16.0/3.0)
|   |   |   |   |   V35 > 48: negative (4.0)
|   |   |   |   V5 > 0.094949
|   |   |   |   |   V16 <= 0.002: negative (24.0/3.0)
|   |   |   |   |   V16 > 0.002: positive (2.0)
```

=== Evaluation on test set (200 cases) ===

Correctly Classified:	**127**	**63.5**	**%**
Incorrectly Classified:	73	36.5	%

=== Confusion Matrix ===

```
 a  b   <- classified as
89 31 |  a = 0
42 38 |  b = 1
```

This part not treated in class – ignore for exam!

An Even Simpler Tree …

This part not treated in class – ignore for exam!

```
V76 = 1: negative (21.0/1.0)
V76 = 2
|   V5 <= 0.090909: negative (156.0/32.0)
|   V5 > 0.090909
|   |   V12 <= 0.009999
|   |   |   V40 <= 0
|   |   |   |   V35 <= 46: positive (37.0/8.0)
|   |   |   |   V35 > 46: negative (14.0/3.0)
|   |   |   V40 > 0
|   |   |   |   V16 <= 0.001
|   |   |   |   |   V31 <= 2.219061: positive (19.0/6.0)
|   |   |   |   |   V31 > 2.219061: negative (24.0/5.0)
|   |   |   |   V16 > 0.001: negative (69.0/10.0)
|   |   V12 > 0.009999: positive (104.0/33.0)
```

=== Evaluation on test set (200 cases) ===

Correctly Classified:	**132**	**66 %**
Incorrectly Classified:	68	34 %

=== Confusion Matrix ===

```
 a  b   ← classified as
89 31 |  a = 0
37 43 |  b = 1
```

➔ Overfitting and Tree Complexity

Is the model (e.g., decision tree) that best fits the training data the best model? NOT NECESSARILY!

Problems:

- Training examples may contain errors ("noise")
- Target concept may not be perfectly describable in given representation
- Greedy heuristic learning algorithm may choose irrelevant tests

  ➔ **OVERFITTING:**
  - Model that tightly fits the training data contains irrelevant features
  - Model is more complex than necessary
  - Model predicts poorly on new, unseen data

Solution:

- Learn simplified models that don't fit the training data exactly
  ➔ MODEL SELECTION
  ➔ *central topic in KV "Machine Learning & Pattern Classification" (SS '15)*

This part not treated in class – ignore for exam!

Summary (Part A)

This part not treated in class – ignore for exam!

- **Learning ability** is a central ingredient of intelligent agents
- Central (possibly even defining) characteristic of learning: always involves **generalisation** (logically speaking: **inductive inference**)
- **Concept learning from classified examples** is one of the best-studied subfields of machine learning
- **Decision trees** are a convenient representation for concepts expressed in a "propositional" attribute-value representation
- ID3 and related algorithms can **efficiently learn decision trees** from large numbers of pre-classified examples
- An important practical application field for concept learning algorithms: **Data Analysis / Data Mining**

Part 9:
Machine Learning

Part 9B:
Learning Simple Action Strategies: Reinforcement Learning
(A Decision-theoretic View of Learning)

Univ.-Prof. Dr. Gerhard Widmer
Department of Computational Perception
Johannes Kepler University Linz

gerhard.widmer@jku.at
http://www.cp.jku.at/people/widmer

Overview of Part B

What is Reinforcement Learning

Reinforcement learning: general formulation

- The learning scenario
- Formal problem statement
- Q-learning algorithm
- Convergence
- Advanced issues

Examples:

- Q-learning in action (demo)
- Learning to play world-class Backgammon: TD-Gammon

Reinforcement Learning

- Learning to choose optimal actions based on experience gained by **acting in an environment**
- Learning by **trial and error**
- Typical scenario:
  Robot / agent learns to achieve goals in an unknown environment

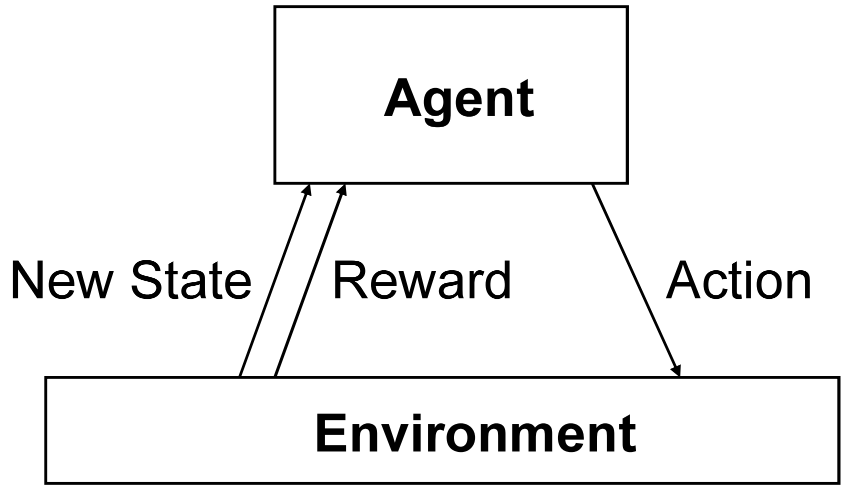
Central problem: Delayed Reward
No immediate classification / feedback available for most actions
(Example: Learning to play chess by playing against a human opponent)
➔ No class labels available for training examples/situations
➔ Cannot use, e.g., decision tree learning algorithm (cf. part A of this document)

➔ **"Temporal Credit Assignment Problem":**
Which of my actions were responsible for the final outcome (success, failure)?
(e.g., in chess: should not assume that all my moves were bad, when I eventually lost the game)

Reinforcement Learning: General Formulation

Scenario:

An **agent** interacts with its **environment**. The agent exists in an environment described by some set of possible **states** $\boldsymbol{S}$. It can perform any of a set of possible **actions** $\boldsymbol{A}$. Each time it performs an action a_t in some state s_t, the agent receives a real-valued **reward** $\boldsymbol{r_t}$ that indicates the immediate value of this state transition (which may also be zero – see "Temporal Credit Assignment Problem"). This produces a sequence of states s_i, actions a_i, and immediate rewards r_i as shown in the figure. The agent's task is to learn a **control policy (action selection strategy)** $\boldsymbol{\pi: S \rightarrow A}$ that maximises the expected sum of these rewards, with future rewards discounted exponentially by their delay.



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \ldots\ldots$$

Goal: Learn to choose actions that maximise

$r_0 + \gamma\, r_1 + \gamma^2 r_2 + \ldots..,$ where $0 \leq \gamma < 1$

Problem Statement

Goal:

Learn **policy ("strategy")** $\pi: S \rightarrow A$
for selecting next action a_t based on current observed state s_t; that is: $\pi(s_t) = a_t$

Criterion:

Policy should produce greatest possible **cumulative reward** over time

Cumulative reward (value) $V_\pi(s_t)$ of policy π from arbitrary initial state s_t:

$$\boxed{V_\pi(s_t) = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \ldots = \sum_{[i=0..\infty]} \gamma^i\, r_{t+i}}$$

$V_\pi(s_t)$ ... "*discounted cumulative reward* achieved by policy π from initial state s_t (with discount factor $\gamma \leq 1$)"

Note:

Other definitions of total reward are possible, e.g.,

- finite horizon reward: $\sum_{[i=0..h]} r_{t+i}$
- average reward: $lim_{[h\rightarrow\infty]}\ 1/h * \sum_{[i=0..h]} r_{t+i}$

Problem Statement (refined)

Goal:

Learn **optimal policy** $\pi^*: S \rightarrow A$
(policy that maximizes $V_\pi(s)$ for all possible starting states s):

$$\pi^* = arg\ max_\pi\ V_\pi(s), \quad \forall\ s$$

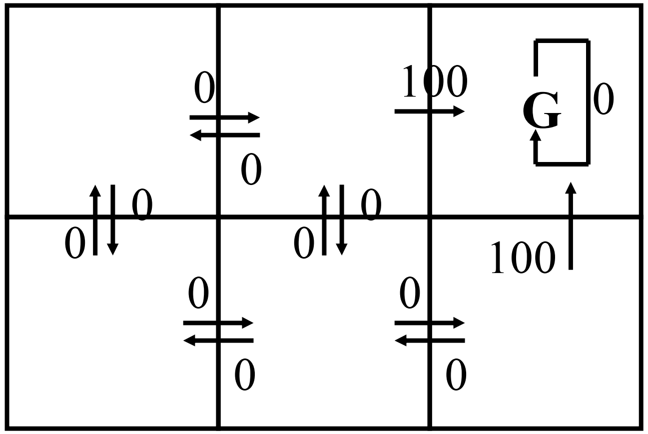
➔ $V^*(s) = V_{\pi^*}(s)$ = discounted cumulative reward produced by optimal strategy π^* , when starting from state s

= maximum discounted cumulative reward that is possible when starting from state s

---

Reminder: the function $arg\ max_{x \in X}\ f(x)$ returns that x^* which maximises $f(x)$

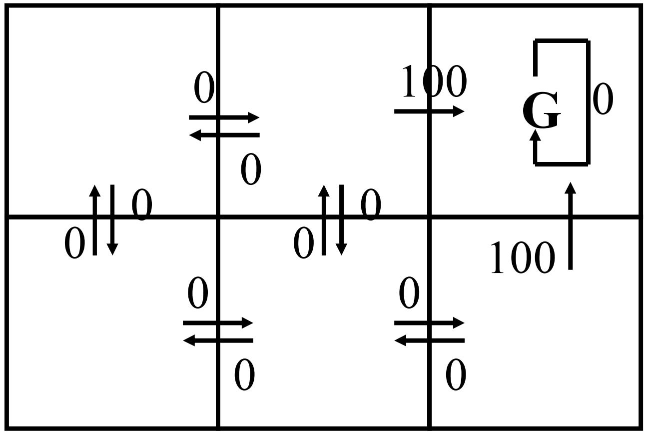
A Very Simple Agent World



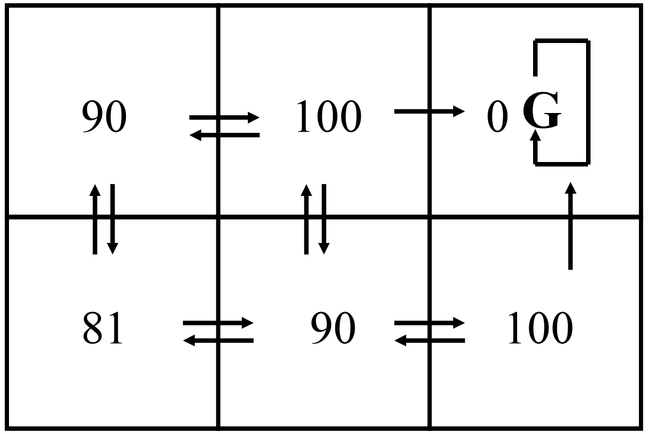
- Squares = set S of possible world states
- Arrows = set A of possible actions and resulting new state s' (***state transition function*** $\delta(s,a) = s'$)
- Numbers attached to arrows: immediate reward $r(s,a)$ produced by taking action a in state s

Important: The agent knows S and A, but not $r(s,a)$!
(in fact, it may not even know S, and it may not know the state transition function $\delta(s,a)$ – i.e., the effects of actions)

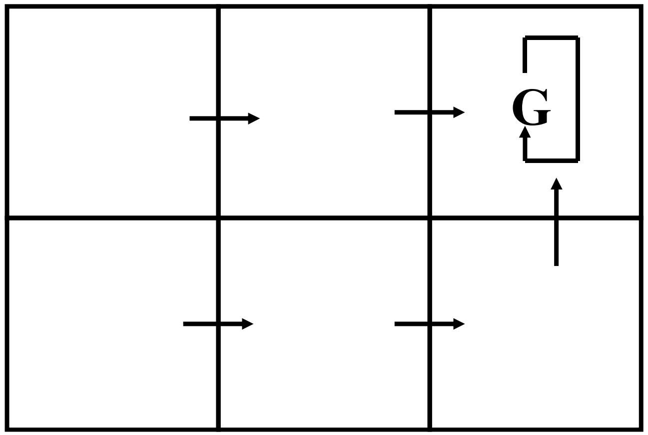
$r(s,a)$ and $V^*(s)$ in Our Very Simple Agent World



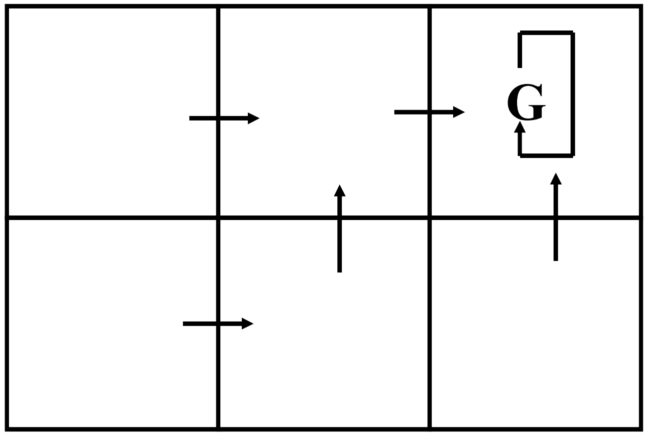
$r(s,a)$ = immediate reward values



$V^*(s)$ values (for $\gamma = 0.9$)
(unknown to agent)



➔ One optimal policy



➔ An alternative policy

Q-Learning

Goal:

Learn optimal policy π^* for a given environment

Problem:

Difficult to learn **action selection function** $\pi^*: S \rightarrow A$ directly (e.g., in the form of a decision tree), because training examples of the form $< s, a >$ (i.e., situations *with known optimal action)* are not available (feedback does not say what the optimal action would have been)

Instead:

Training data (feedback) is a sequence of immediate rewards $r(s_i, a_i)$ for $i=0,1,2,...$

➔ **Possible Solution (1):**

Learn **numerical evaluation function** $E: S \rightarrow R$ instead (i.e., function that evaluates the value of being in state s = that estimates the ***reward achievable*** from s)

➔ **Note:** The optimal / most accurate E would of course be V^* (the truly achievable maximum reward) …

Q-Learning

Optimal evaluation function $V^*: S \rightarrow R$

- Agent should prefer state s_1 over state s_2 whenever $V^*(s_1) > V^*(s_2)$
- Optimal action in state s is action a that maximises immediate reward $r(s,a)$ plus V^* of immediate successor state (discounted by γ):

$$\pi^*(s) = \arg\max_a \left(r(s,a) + \gamma\, V^*(\delta(s,a))\right)$$

where $\delta(s,a)$ = new state resulting from applying action a to state s
(δ = *'state transition function'*)

➔ **Problem:**
$V^*(s)$ can be used to determine π^* only if reward function r and state transition function δ are known (i.e., if the agent can predict the consequences of an action, or tentatively try it out to observe the outcome and then undo it)

➔ **Solution (2):**
Learn **evaluation function for pairs (state, action)** instead: $Q: S \times A \rightarrow R$
(i.e., function that evaluates the promise of taking action a *in state* s)

Q-Learning

Definition of optimal Q^*: $S \times A \rightarrow R$ **:**

$$Q^*(s,a) = r(s,a) + \gamma\, V^*(\delta(s,a))$$

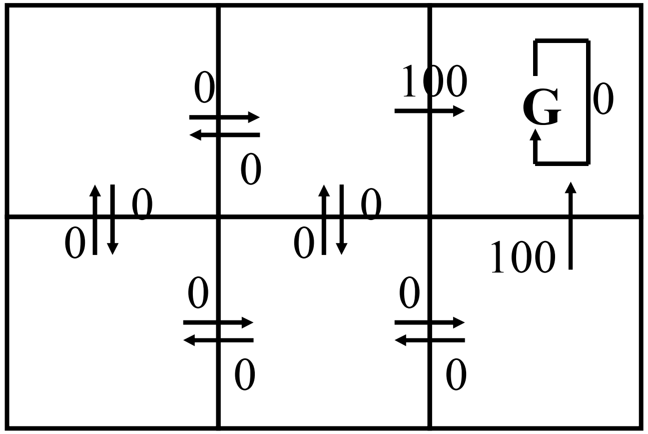
(max. achievable reward if we start with action a)

➔ $\pi^*(s) = arg\ max_a\ (r(s,a) + \gamma\, V^*(\delta(s,a)))$

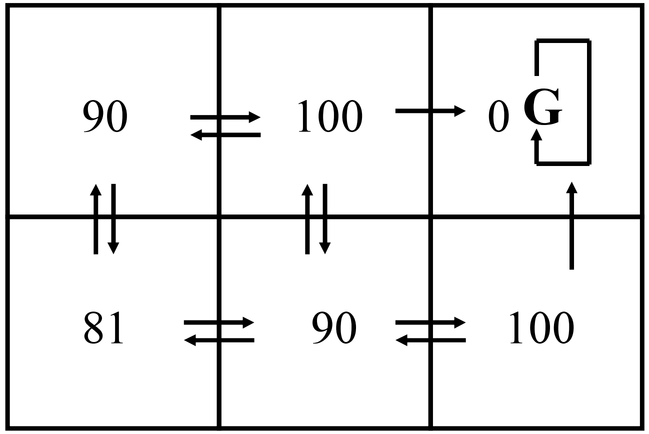
$= arg\ max_a\ Q^*(s,a)$!

➔ If the learner can learn function Q^* (or some good approximation Q of it) instead of function V^*, it will be able to select optimal actions without knowledge of functions r and δ

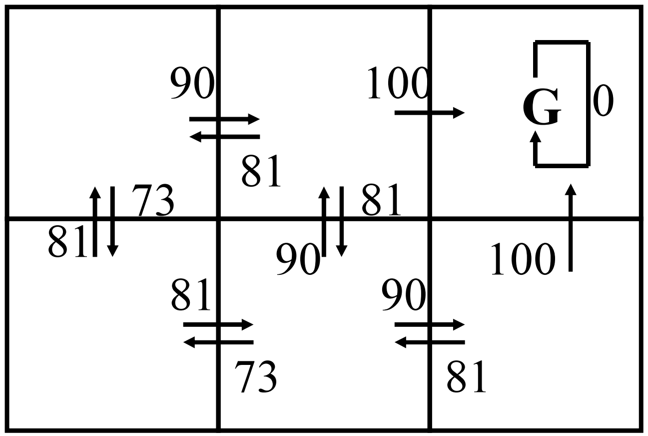
A Simple Example



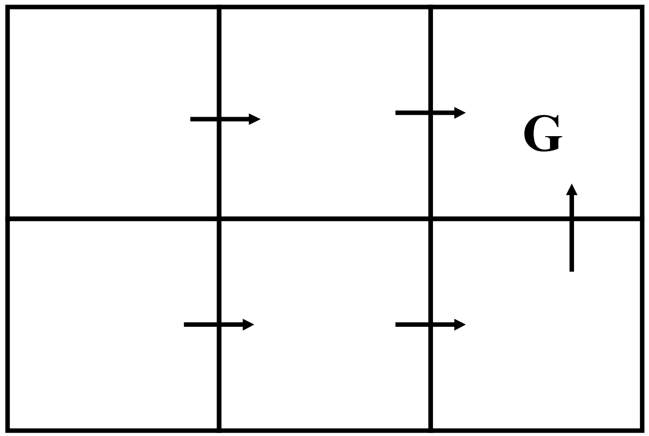
$r(s,a)$ (immediate reward) values



$V^*(s)$ values (for $\gamma = 0.9$)



$Q^*(s,a)$ values



One optimal policy

Algorithm for Learning an Approximate Evaluation Function Q

Key Problem:

Would need training examples of the form $< s, a, q^* >$ for learning Q , but we don't know the true q^* values for learning.

Need to ***estimate*** training values for Q, given only a sequence of immediate rewards r_i spread out over time

Note:

Close relation between Q^* and V^*:

$$Q^*(s,a) = r(s,a) + \gamma\ V^*(\delta(s,a)) \qquad \text{[Def.]}$$

$$V^*(s) = max_{a'}\ Q^*(s,a')$$

$$\Rightarrow \quad Q^*(s,a) = r(s,a) + \gamma\ max_{a'}\ Q^*(\delta(s,a),\ a')$$

➔ **Key Idea: Iterative Approximation:**

Assume we already have a 'reasonable' approximation for $Q^*(\delta(s,a), \cdot)$

Use current approximation Q of successor state $\delta(s,a)$

to improve approximation of Q for action a in current state s

Effect: back-propagation of information through sequences of actions

Algorithm for Learning Q

Simplest possible representation of hypothesis (approximation) Q :

Large table with separate entry for each state-action pair $<s,a>$

Algorithm:

For each s, a initialize the table entry $Q(s,a)$ to zero;
Observe the current state s;
Do forever:

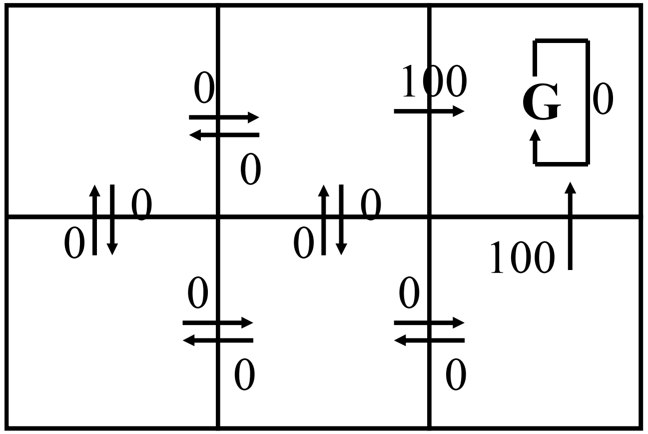
- Select an action a and execute it
- Receive immediate reward $r(s,a)$
- Observe the new state $\delta(s,a)$
- Update the table entry for $Q(s,a)$ as follows:

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

- $s \leftarrow \delta(s,a)$

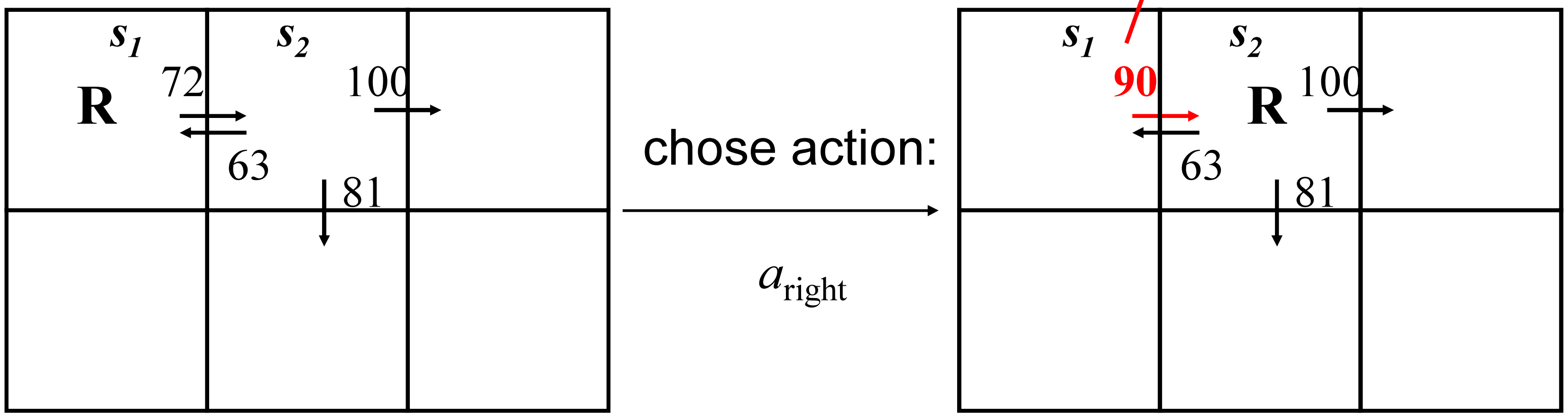
A Simple Example

Artificial world defined by functions δ and r:



$$Q(s_1,a_{right}) \leftarrow r + \gamma \max_{a'} Q(s_2,a')$$
$$= 0 + 0.9 \max\{63, 81, 100\}$$
$$= 90$$

One learning step (γ = 0.9):



Convergence

Theorem: Convergence of Q learning for deterministic Markov decision processes (MDPs)*

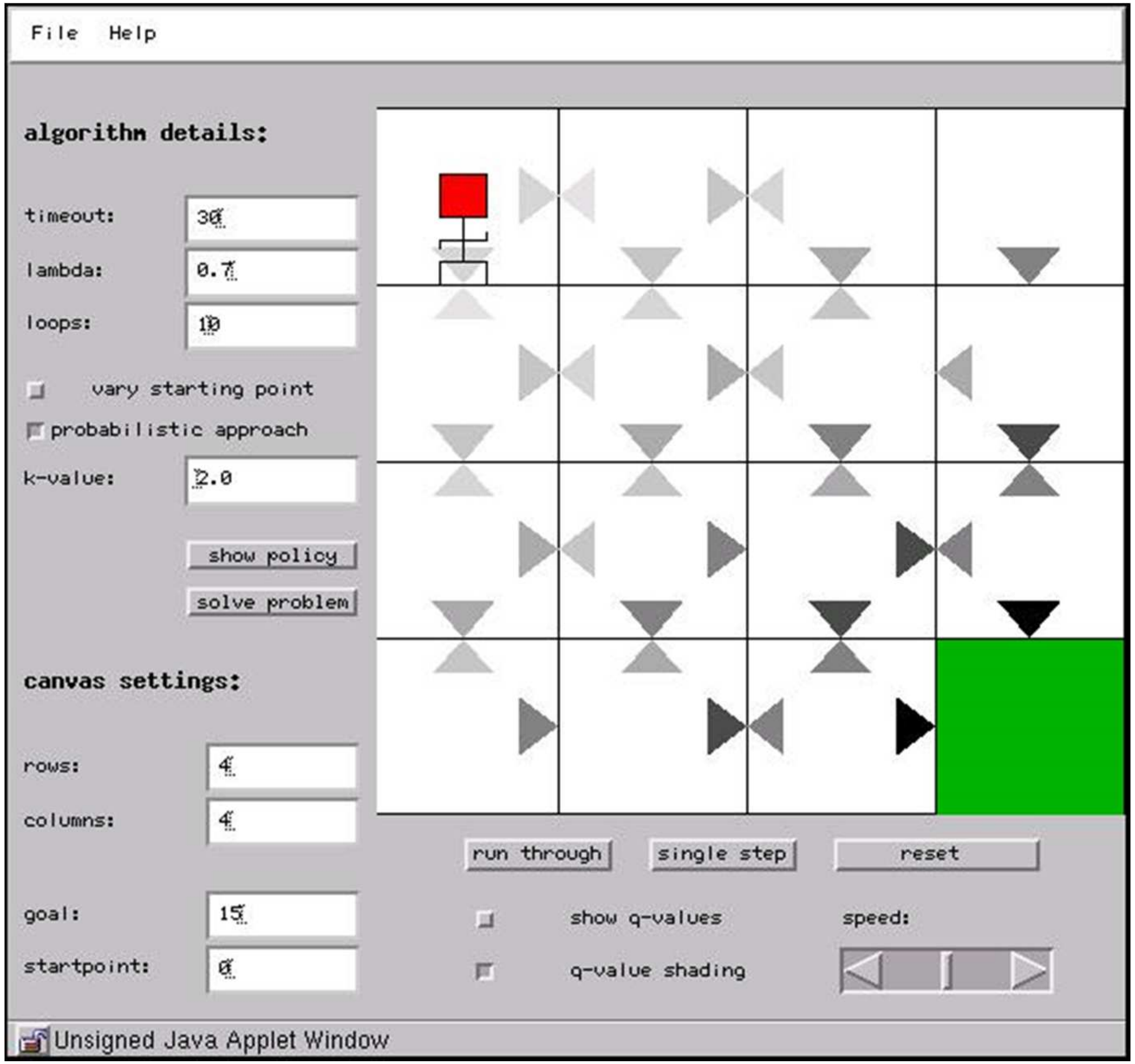
- Consider a Q learning agent in a deterministic MDP with bounded rewards $(\forall\ s,a)\ |r(s,a)| < c$.
- The agent uses the training rule $Q(s,a) = r(s,a) + \gamma\ max_{a'}\ Q(\delta(s,a),\ a')$,
- initialises its table $Q(s,a)$ to arbitrary finite values, and
- uses a discount factor γ such that $0 \leq \gamma < 1$.

- Let $Q_n(s,a)$ denote the agent's hypothesis $Q(s,a)$ following the n^{th} update.

➔ If each state-action pair is visited infinitely often, !
then $Q_n(s,a)$ converges to $Q^*(s,a)$ as $n \rightarrow \infty$, for all $s,\ a$.

*) ***Deterministic Markov process***: a world where the next state and reward depend only (and deterministically) on the previous state and action.

Live Demonstration



Run demo ...

Advanced Issues

- **Introducing generalisation ability**
  – using features and function approximators (e.g., neural nets, ...) to represent states and Q (instead of a huge table that only remembers actually encountered situations)

- **How to select the next action? (always follow current Q hypothesis, or try alternatives?)**
  – "exploitation vs. exploration trade-off"

- **Improving learning speed**
  – remembering and replaying entire episodes

- **Learning in nondeterministic worlds**

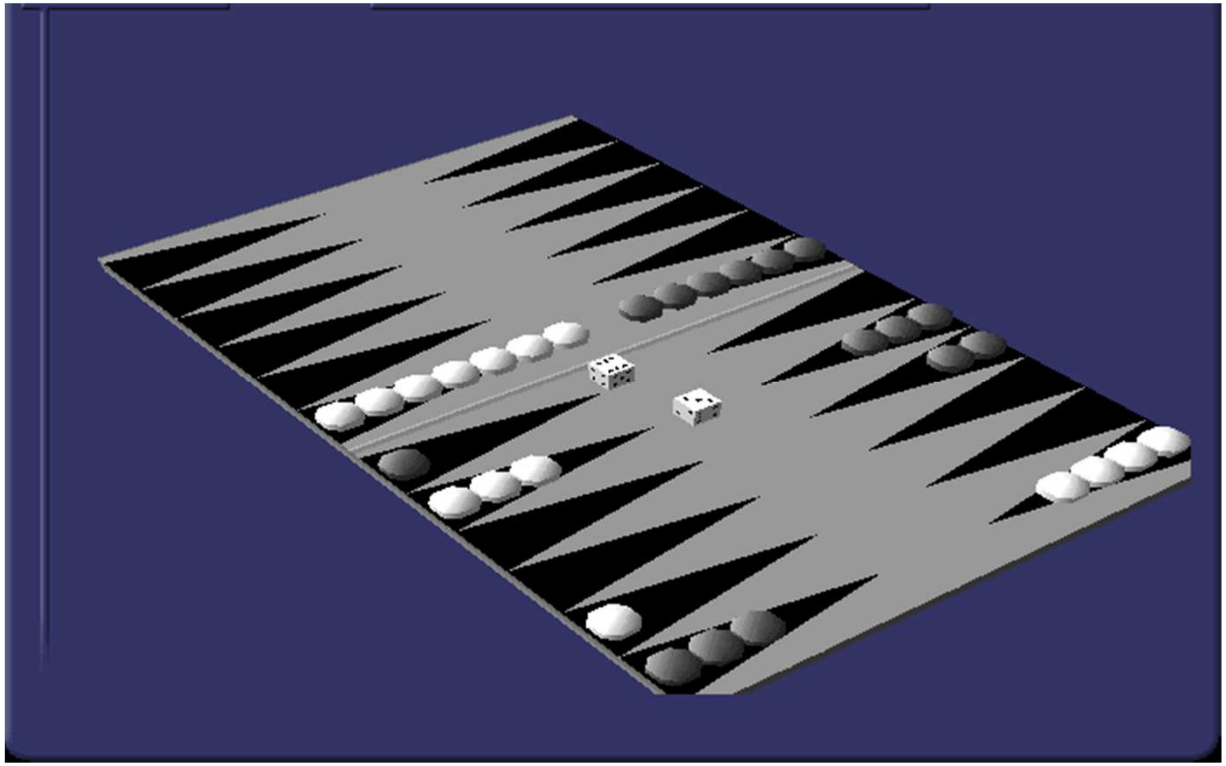
- **Accepting advice from teacher**

- **Hierarchical reinforcement learning**

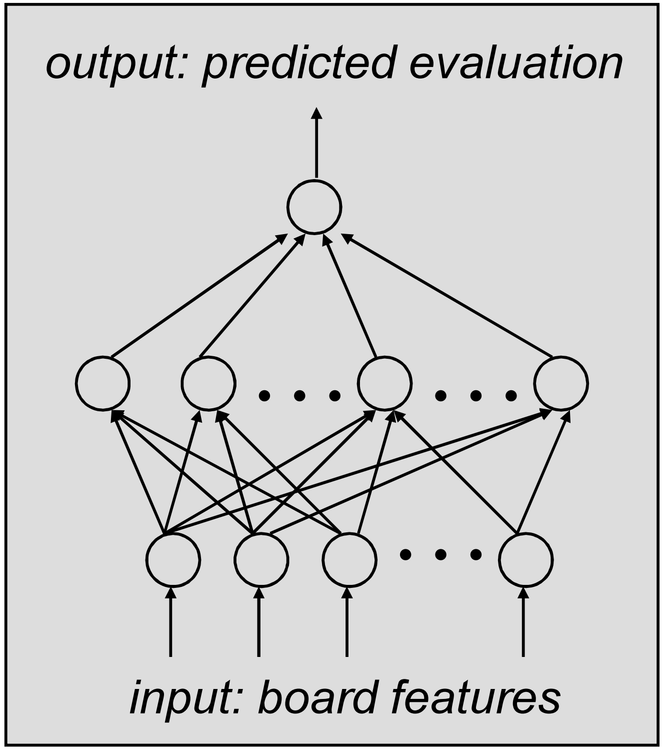
- **.....**

Reinforcement Learning in Game Playing: TD-Gammon (Tesauro, 1995)

- World-class backgammon playing program
- Learned to play only from ***playing against itself (!)***
- 'TD' ... *'temporal difference learning'* (generalisation of Q-learning)



TD-Gammon



Target function:
Evaluation function ***E: BoardStates*** $\rightarrow$ ***R***

Representation of target function *E* :
Feed-forward neural network with one hidden layer
(instead of ***Q*** table ➔ forces learner to generalise)

Representation of board states (= input to NN):
Raw features (numbers of white and black checkers at each location) + some predefined higher-level features (e.g., strength of a blockade)

Output of neural network:
Numerical evaluation of estimated quality ***E'*** of a board state

Learning algorithm:
TD(λ), generalisation of ***Q*** learning

TD-Gammon: Results

- Excellent play against world-class human grandmasters (including some former world champions)

Results of testing TD-Gammon in play against world-class human opponents.
Version 1.0 used 1-ply search for move selection; versions 2.0 and 2.1 used 2-ply search.
Version 2.0 had 40 hidden units; versions 1.0 and 2.1 had 80 hidden units.

Program	*Training Games*	*Opponents*	*Results*
TDG 1.0	300,000	Robertie, Davis, Magriel	-13 pts / 51 games (-.25 ppg)
TDG 2.0	800,000	Goulding, Woolsey, Snellings, Russell, Sylvester	-7 pts / 38 games (-0.18 ppg)
TDG 2.1	1,500,000	Robertie	-1 pt / 40 games (-0.02 ppg)

- In a few cases, has changed human experts' judgement of the best move to make in particular situations

Summary (Part B)

- Main problem in learning the utility of actions: **delayed reward** and **credit assignment problem**
- Basic idea in Reinforcement Learning for dealing with this problem: **back-propagation of information** through sequences of actions
- **Q-Learning:** function Q represents expected cumulated reward achievable from state s, given that first action is a
- **Simplest (too simple)** form of Q learning: Q realised as **table**
- In realistic applications, **generalisation** is required
  ➔ use of **other types of models** for representing Q
- Many **other forms of reinforcement** learning have been studied
- Reinforcement learning used in **many applications** (on-line optimisation …)