Machine Learning 2D5362

Lecture 1:

Introduction to Machine Learning

Machine Learning

- Date/Time:
  Tuesday ???
  Thursday 13.30
- Location:
  BB2 ?
- Course requirements:
  active participation
  homework assignments
  course project
- Credits:
  3-5 credits depending on course project
- Course webpage:
  http://www.nada.kth.se/~hoffmann/ml.html

Course Material

Textbook (recommended):

- Machine Learning

  Tom M. Mitchell, McGraw Hill,1997

  ISBN: 0-07-042807-7 (available as paperback)

Further readings:

- An Introduction to Genetic Algorithms

  Melanie Mitchell, MIT Press, 1996
- Reinforcement Learning – An Introduction

  Richard Sutton, MIT Press, 1998
- Selected publications:

  check course webpage

Course Overview

- Introduction to machine learning
- Concept learners
- Decision tree learning
- Neural networks
- Evolutionary algorithms
- Instance based learning
- Reinforcement learning
- Machine learning in robotics

Software Packages & Datasets

- MLC++
  - Machine learning library in C++
  - http://www.sig.com/Technology/mlc
- GALIB
  - MIT GALib in C++
  - http://lancet.mit.edu/ga
- UCI
  - Machine Learning Data Repository UC Irvine
  - http://www.ics.uci.edu/~mlearn/ML/Repository.html

Possible Course Projects

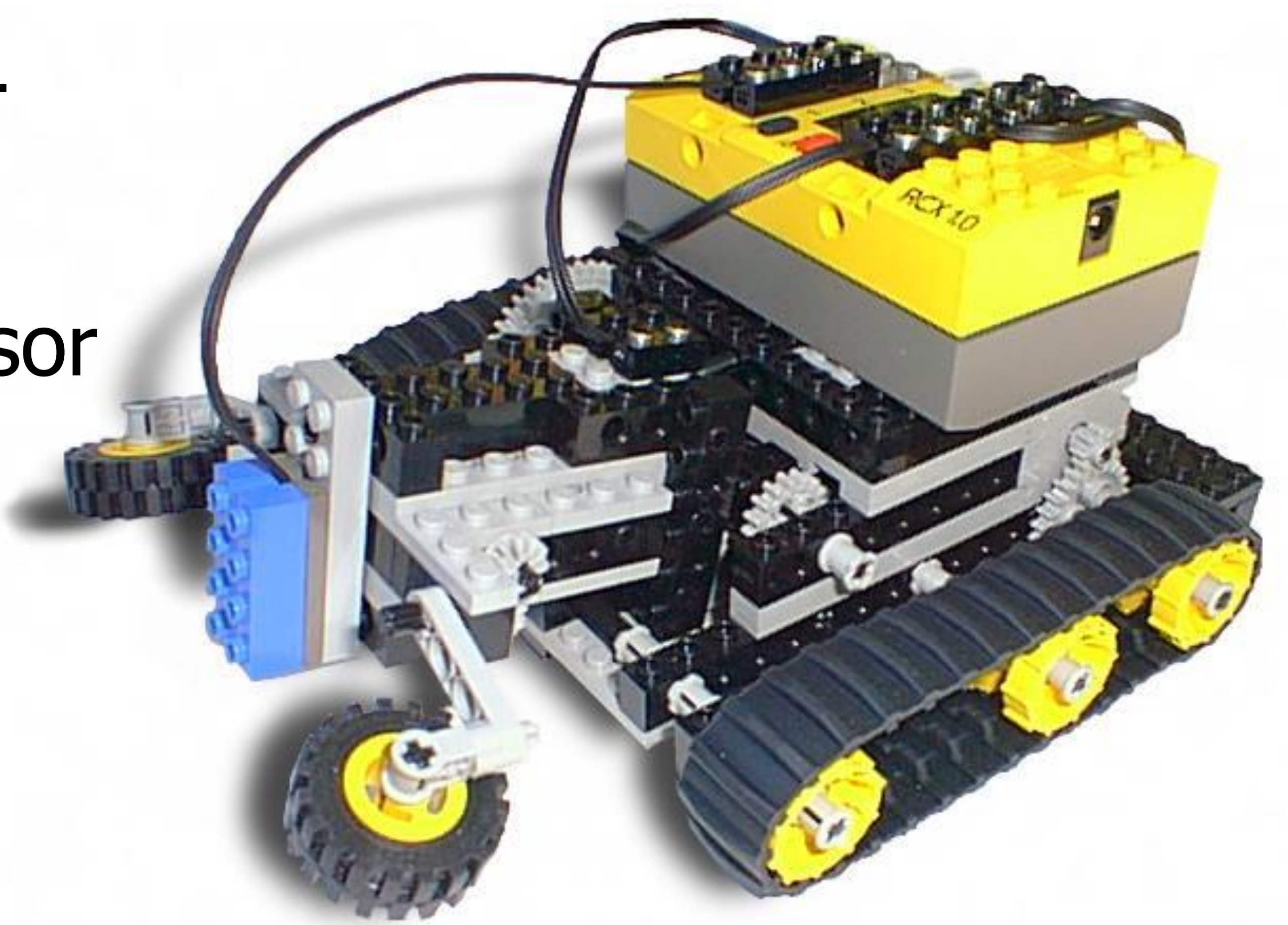
- Apply machine learning techniques to your own problem e.g. classification, clustering, data modeling, object recognition
- Investigating combining multiple classifiers
- Comparing different approaches in genetic fuzzy systems
- Learning robotic behaviors using evolutionary techniques or reinforcement learning
  - LEGO Mindstorm
  - Scout

Scout Robots

- 16 Sonar sensors
- Laser range scanner
- Odometry
- Differential drive
- Simulator
- API in C

LEGO Mindstorms

- Touch sensor
- Light sensor
- Rotation sensor
- Video cam
- Motors



Learning & Adaptation

- ”Modification of a behavioral tendency by expertise.” (Webster 1984)
- ”A learning machine, broadly defined is any device whose actions are influenced by past experiences.” (Nilsson 1965)
- ”Any change in a system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population.” (Simon 1983)
- ”An improvement in information processing ability that results from information processing activity.” (Tanimoto 1990)

Learning

Definition:

A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience.

Disciplines relevant to ML

- Artificial intelligence
- Bayesian methods
- Control theory
- Information theory
- Computational complexity theory
- Philosophy
- Psychology and neurobiology
- Statistics

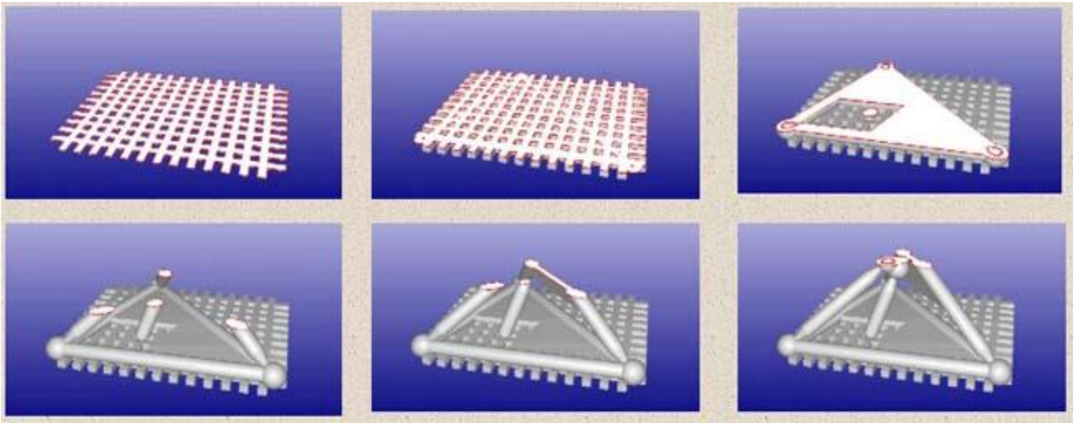
Applications of ML

- Learning to recognize spoken words
  - SPHINX (Lee 1989)
- Learning to drive an autonomous vehicle
  - ALVINN (Pomerleau 1989)
- Learning to classify celestial objects
  - (Fayyad et al 1995)
- Learning to play world-class backgammon
  - TD-GAMMON (Tesauro 1992)
- Designing the morphology and control structure of electro-mechanical artefacts
  - GOLEM (Lipton, Pollock 2000)

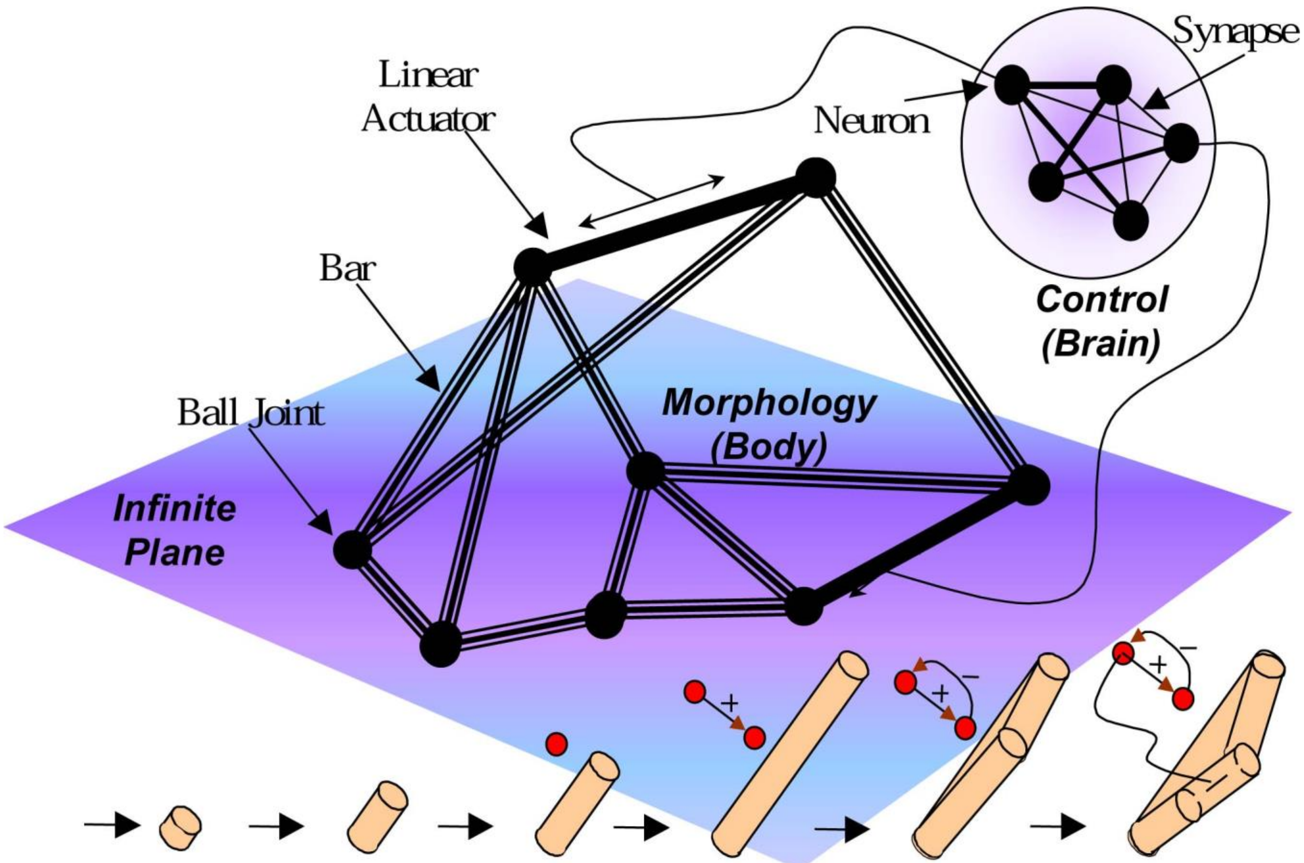
Artificial Life

- GOLEM Project (Nature: Lipson, Pollack 2000)

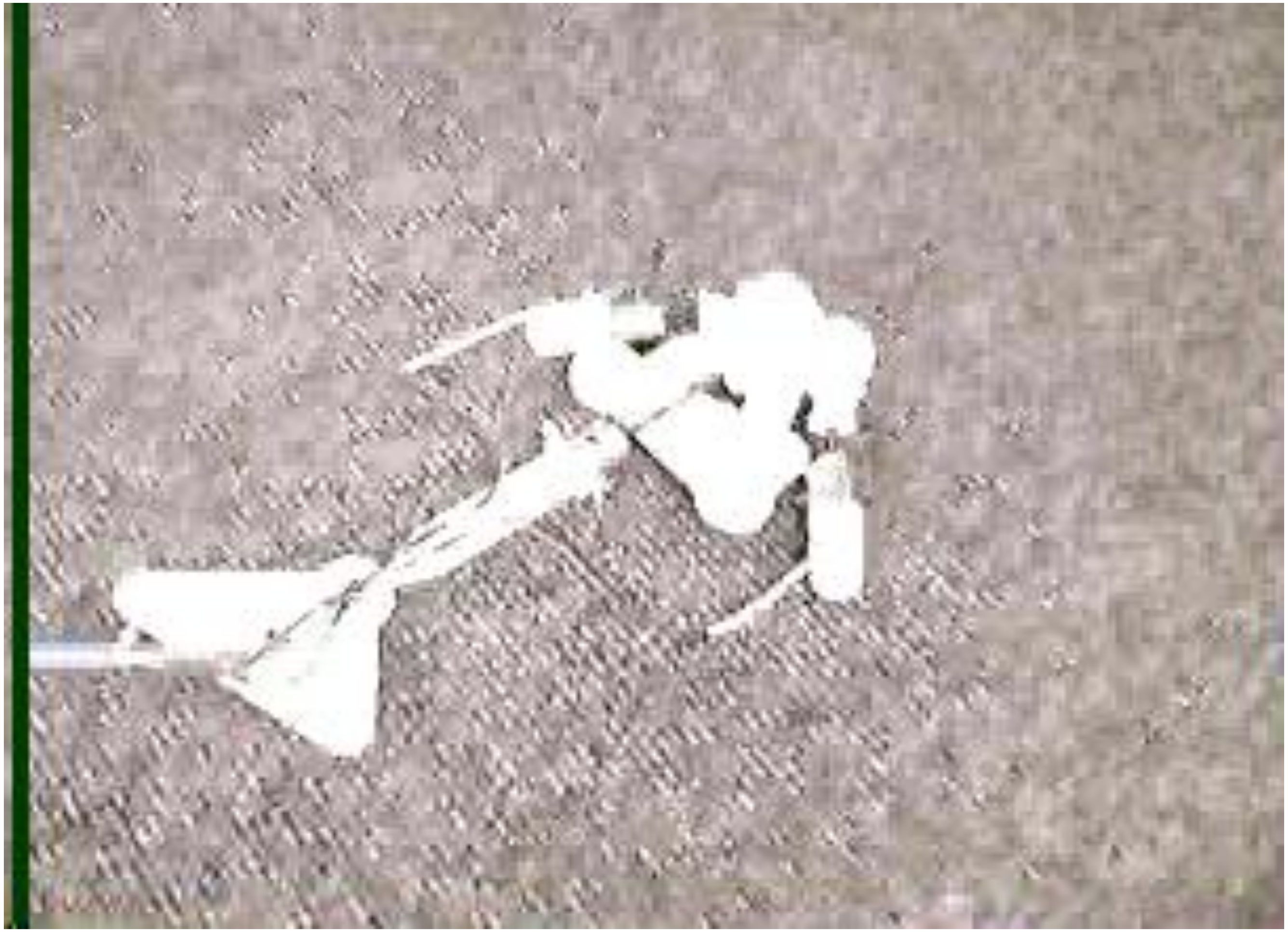
  http://golem03.cs-i.brandeis.edu/index.html
- Evolve simple electromechanical locomotion machines from basic building blocks (bars, acuators, artificial neurons) in a simulation of the physical world (gravity, friction).
- The individuals that demonstrate the best locomotion ability are fabricated through rapid prototyping technology.



Evolvable Robot

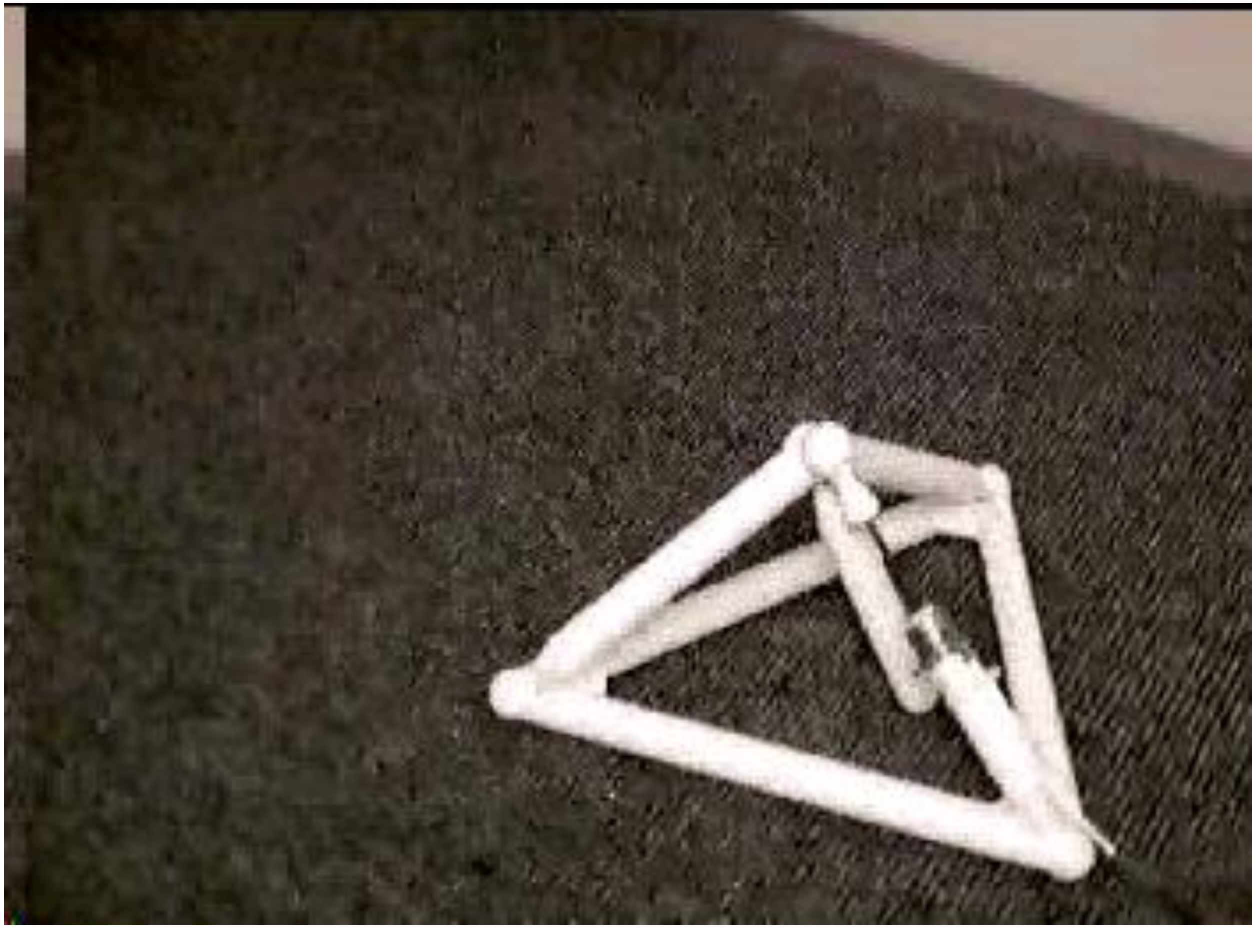


Arrow



Ratchet

Tetra

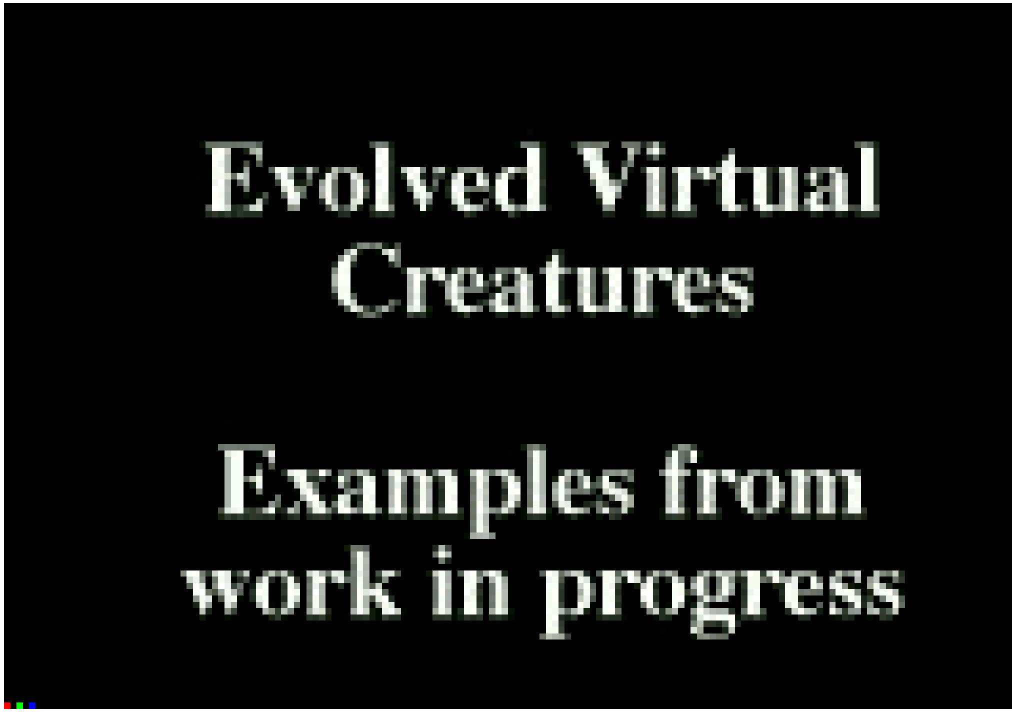


Evolved Creatures

Evolved creatures: Sims (1994)

http://genarts.com/karl/evolved-virtual-creatures.html

Darwinian evolution of virtual block creatures for swimming, jumping, following, competing for a block



Learning Problem

Learning: improving with experience at some task

- Improve over task T
- With respect to performance measure P
- Based on experience E

Example: Learn to play checkers:

- T: play checkers
- P: percentage of games won in a tournament
- E: opportunity to play against itself

Learning to play checkers

- T: play checkers
- P: percentage of games won
- What experience?
- What exactly should be learned?
- How shall it be represented?
- What specific algorithm to learn it?

Type of Training Experience

- Direct or indirect?
  - Direct: board state -> correct move
  - Indirect: outcome of a complete game
  - Credit assignment problem
- Teacher or not ?
  - Teacher selects board states
  - Learner can select board states
- Is training experience representative of performance goal?
  - Training playing against itself
  - Performance evaluated playing against world champion

Choose Target Function

- ChooseMove : $B \rightarrow M$: board state $\rightarrow$ move
  - Maps a legal board state to a legal move
- Evaluate : $B \rightarrow V$: board state $\rightarrow$ board value
  - Assigns a numerical score to any given board state, such that better board states obtain a higher score
  - Select the best move by evaluating all successor states of legal moves and pick the one with the maximal score

Possible Definition of Target Function

- If b is a final board state that is won then V(b) = 100
- If b is a final board state that is lost then V(b) = -100
- If b is a final board state that is drawn then V(b)=0
- If b is not a final board state, then V(b)=V(b’), where b’ is the best final board state that can be achieved starting from b and playing optimally until the end of the game.
- Gives correct values but is not operational

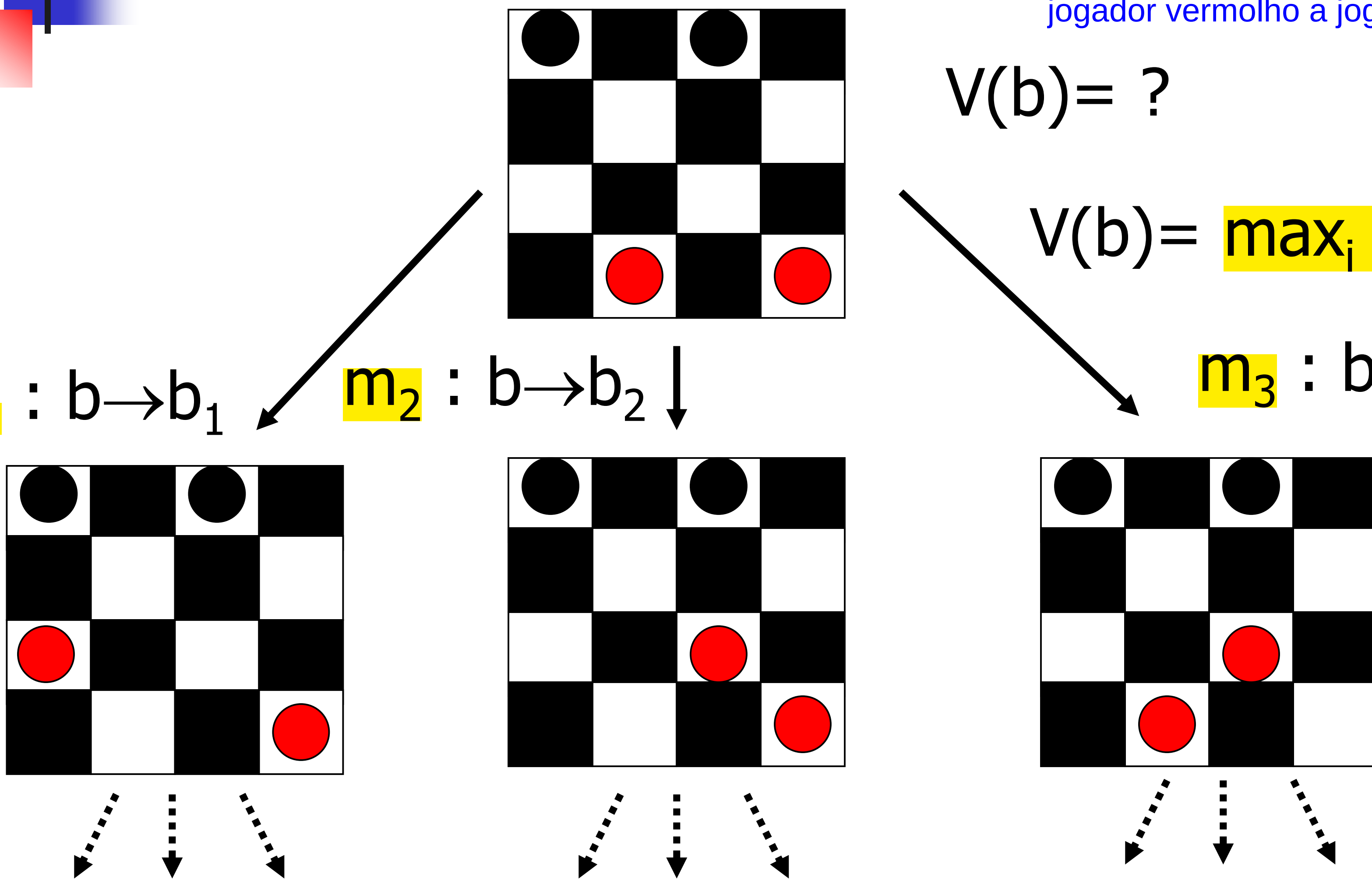
State Space Search

eu sou vermelho

jogador vermolho a jogar

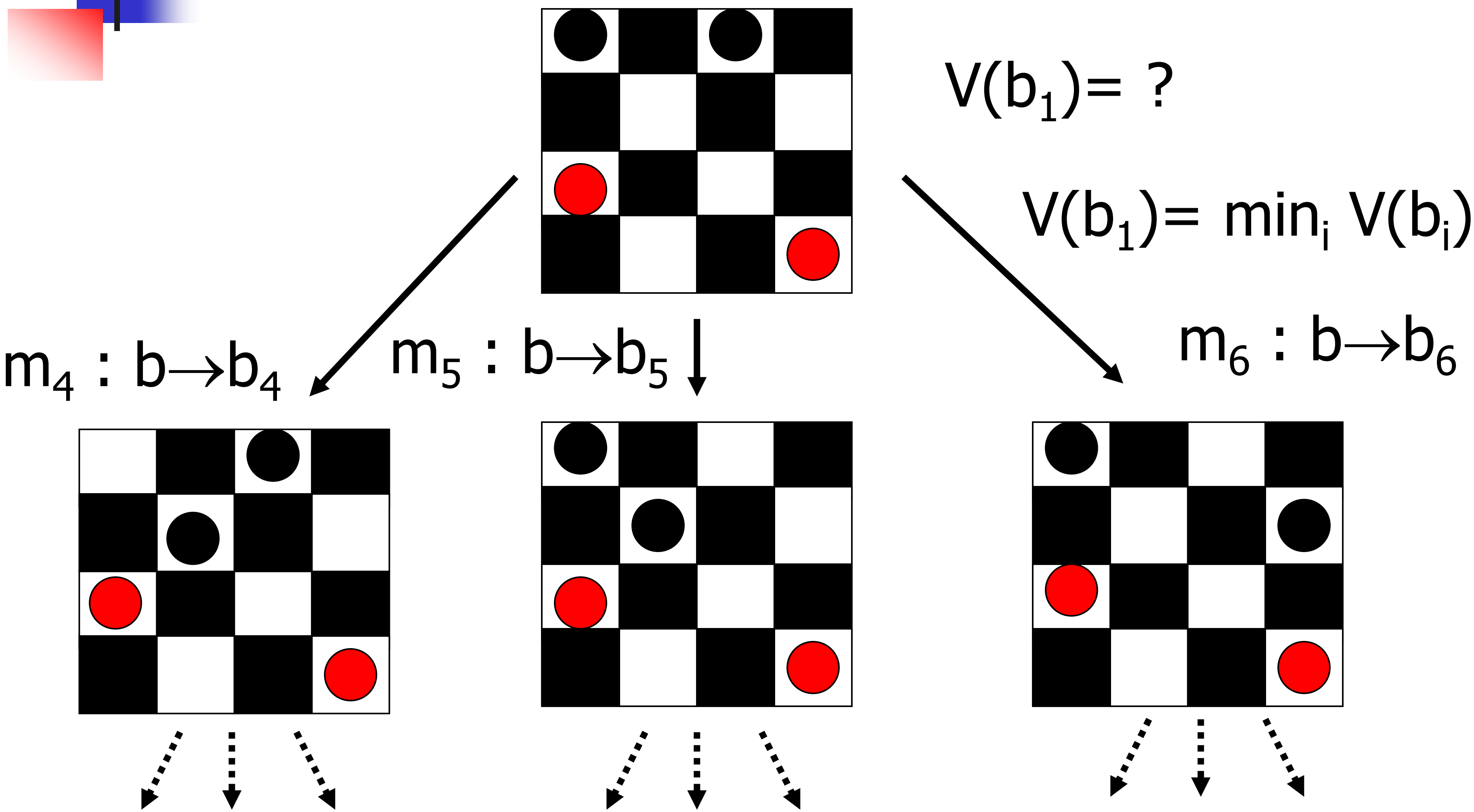
$V(b) = ?$

$V(b) = \max_i V(b_i)$

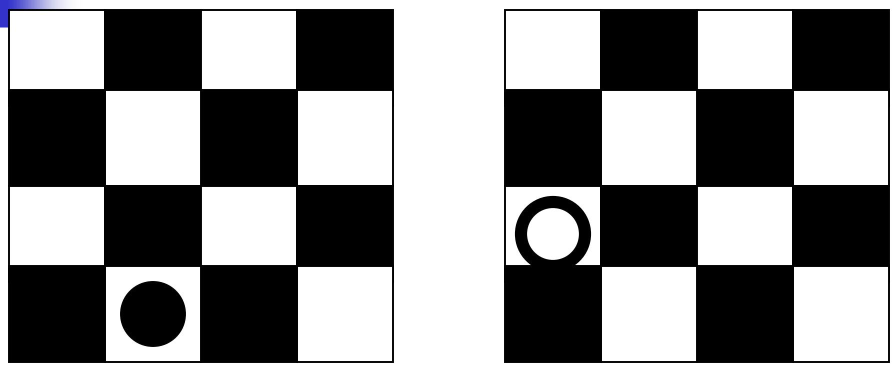


$m_1 : b \rightarrow b_1$ $m_2 : b \rightarrow b_2$ $m_3 : b \rightarrow b_3$

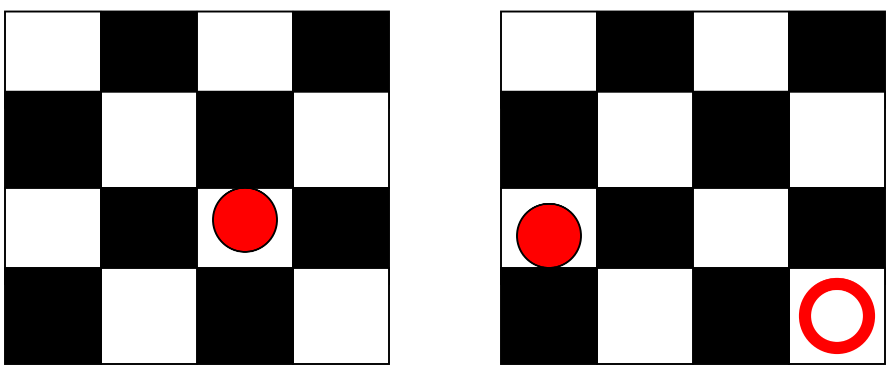
State Space Search



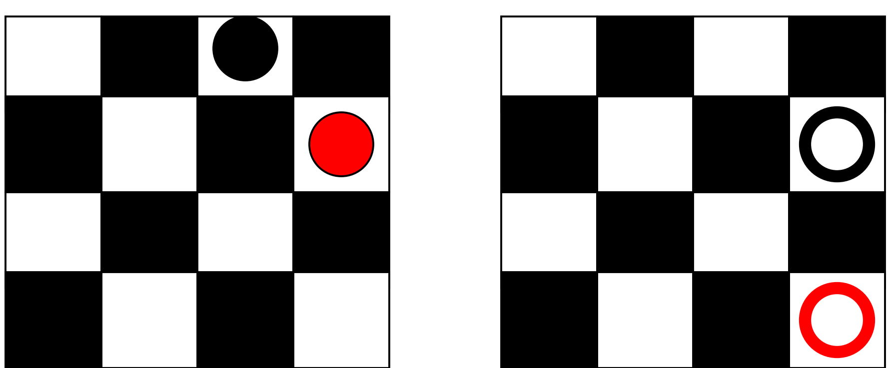
Final Board States



Black wins: V(b)=-100

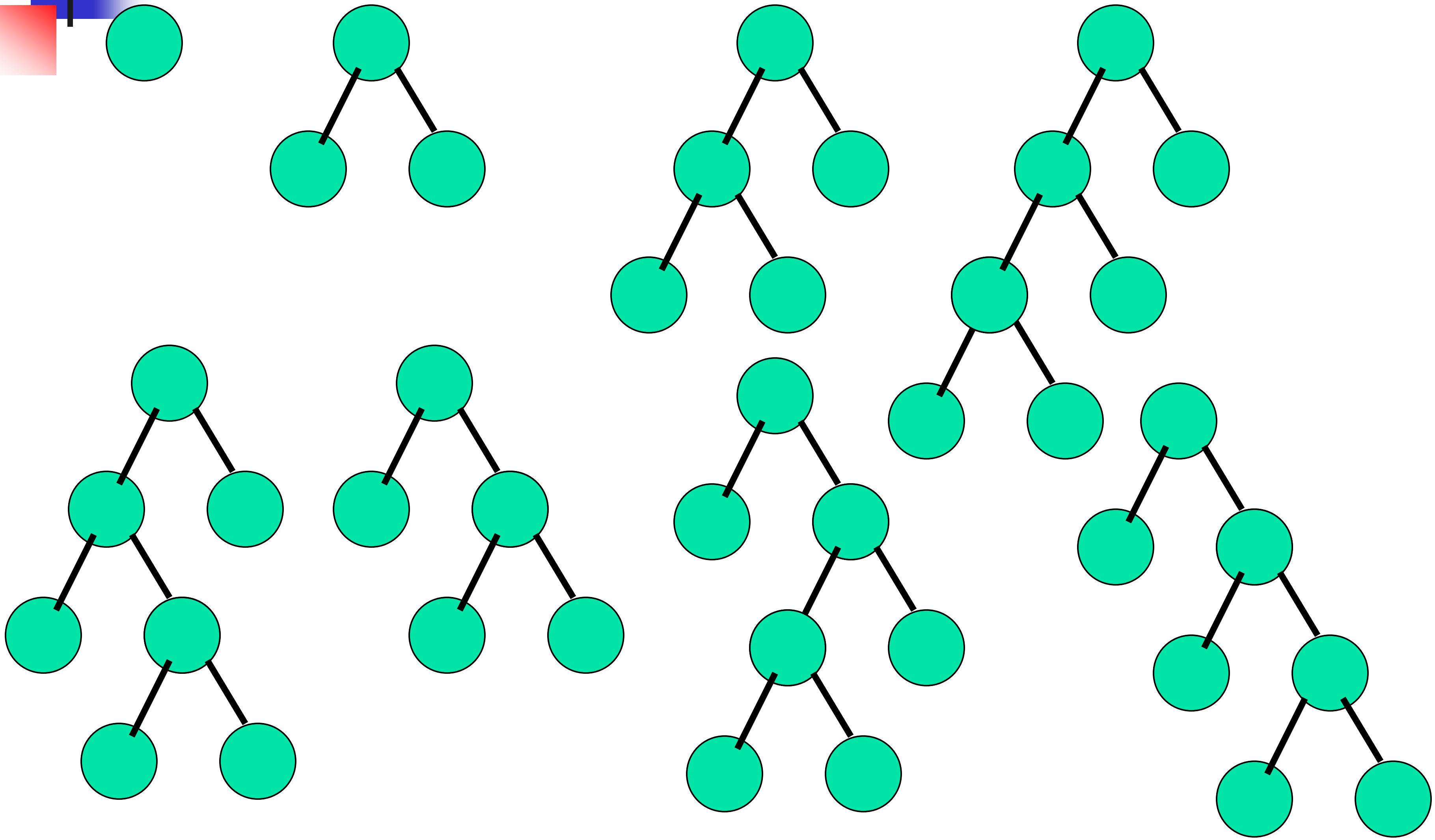


Red wins: V(b)=100

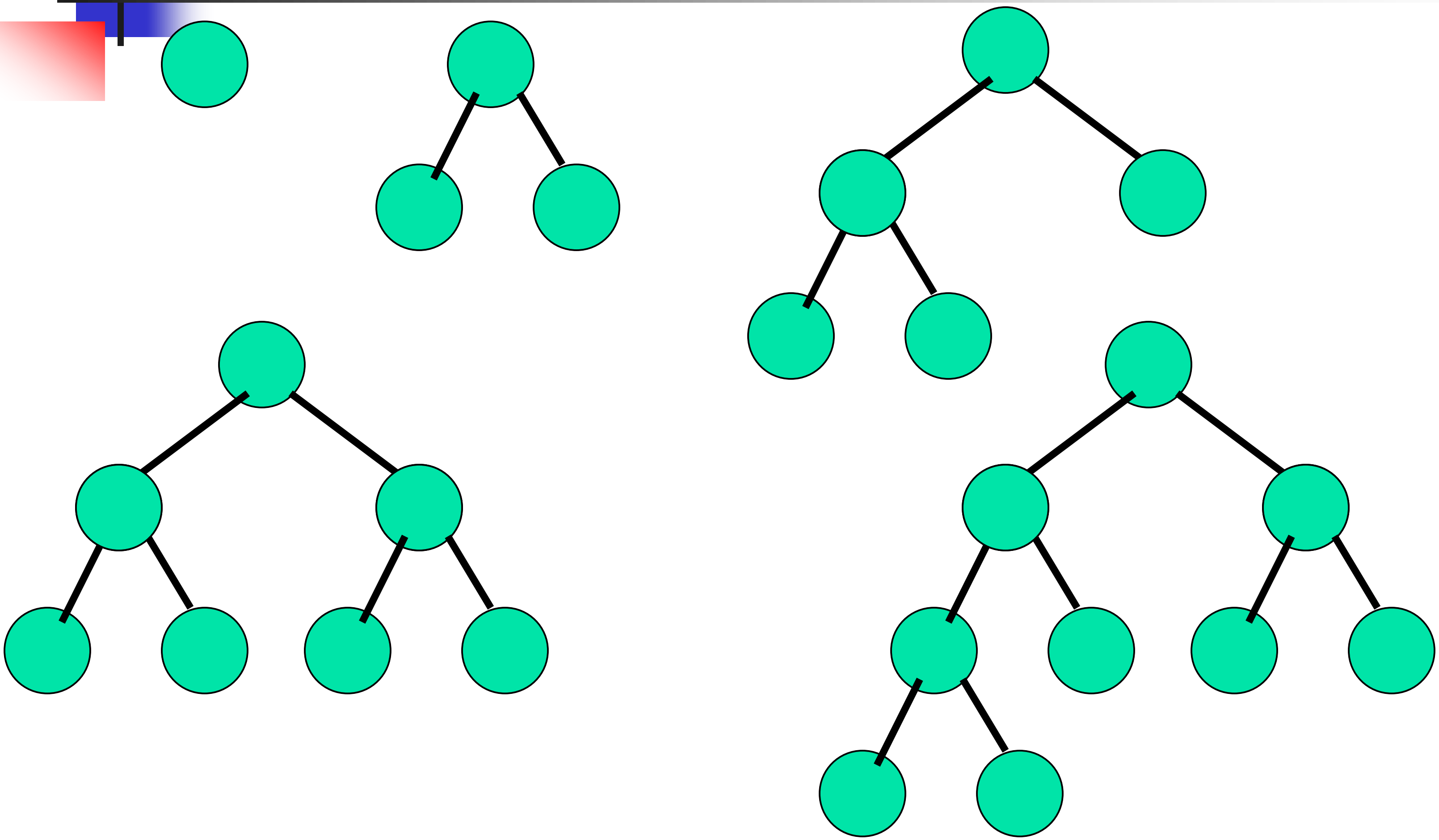


draw: V(b)=0

Depth-First Search

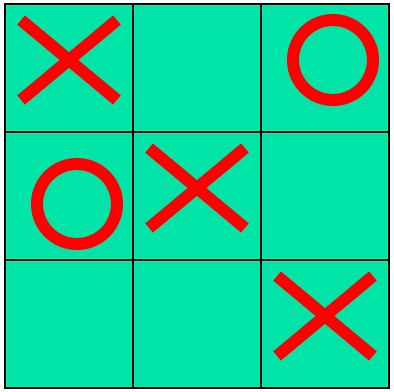


Breadth-First Search



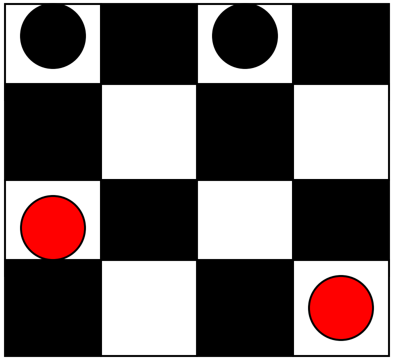
Number of Board States

Tic-Tac-Toe:



#board states < 9!/(5! 4!) + 9!/(1! 4! 4!) + …
… + 9!/(2! 4! 3!) + … 9 = 6045

4 x 4 checkers: (no queens)



#board states = ?

#board states < 8x7x6x5*2^2/(2!*2!) = 1680

Regular checkers (8x8 board, 8 pieces each)

#board states < 32!*2^{16}/(8! * 8! * 16!) = 5.07*10^{17}

Choose Representation of Target Function

- Table look-up
- Collection of rules
- Neural networks
- Polynomial function of board features
- Trade-off in choosing an expressive representation:
  - Approximation accuracy
  - Number of training examples to learn the target function

Representation of Target Function

$V(b)=\omega_0 + \omega_1 bp(b) + \omega_2 rp(b) + \omega_3 bk(b) + \omega_4 rk(b) + \omega_5 bt(b) + \omega_6 rt(b)$

- bp(b): #black pieces
- rb(b): #red pieces
- bk(b): #black kings
- rk(b): #red kings
- bt(b): #red pieces threatened by black
- rt(b): #black pieces threatened by red

Obtaining Training Examples

- $V(b)$: true target function

  -100
  100
  0

- $V'(b)$: learned target function
- $V_{train}(b)$: training value
- Rule for estimating training values:
- $V_{train}(b) \leftarrow V'(\text{Successor}(b))$

Choose Weight Training Rule

LMS weight update rule:

- Select a training example b at random

1. Compute error(b)

actual

anterior

$$error(b) = V_{train}(b) - V'(b)$$

2. For each board feature fi, update weight $\omega_i \leftarrow \omega_i + \eta\, f_i\, error(b)$

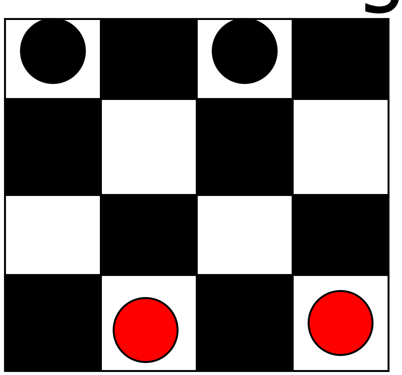
η : learning rate approx. 0.1

escolho maximo

Example: 4x4 checkers

$V(b) = \omega_0 + \omega_1 rp(b) + \omega_2 bp(b)$

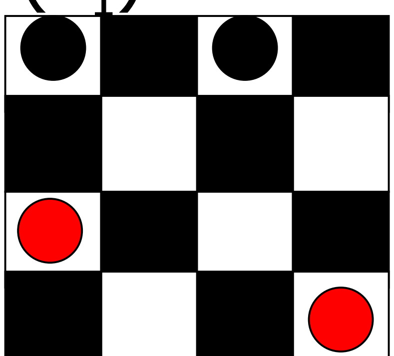
Initial weights: $\omega_0 = -10$, $\omega_1 = 75$, $\omega_2 = -60$



$V(b_0) = \omega_0 + \omega_1 * 2 + \omega_2 * 2 = 20$

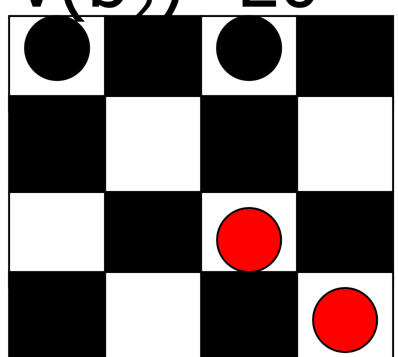
m$_1$: $b \rightarrow b_1$

$V(b_1) = 20$



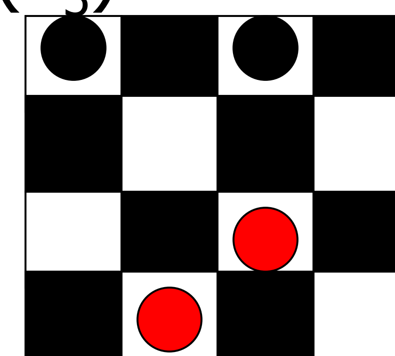
m$_2$: $b \rightarrow b_2$

$V(b_2) = 20$

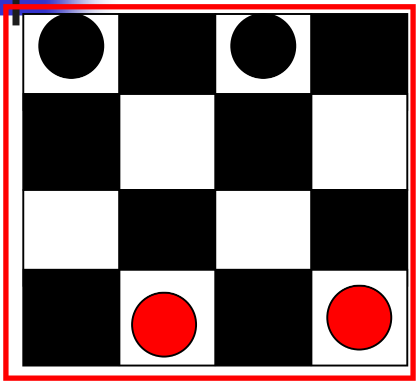


m$_3$: $b \rightarrow b_3$

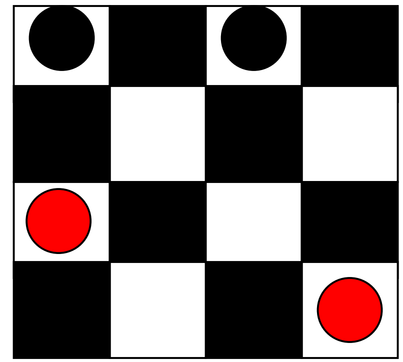
$V(b_3) = 20$



Example 4x4 checkers



$V(b_0)=20$



$V(b_1)=20$

1. Compute $\text{error}(b_0) = V_{train}(b) - V(b_0) = V(b_1) - V(b_0) = 0$
2. For each board feature fi, update weight

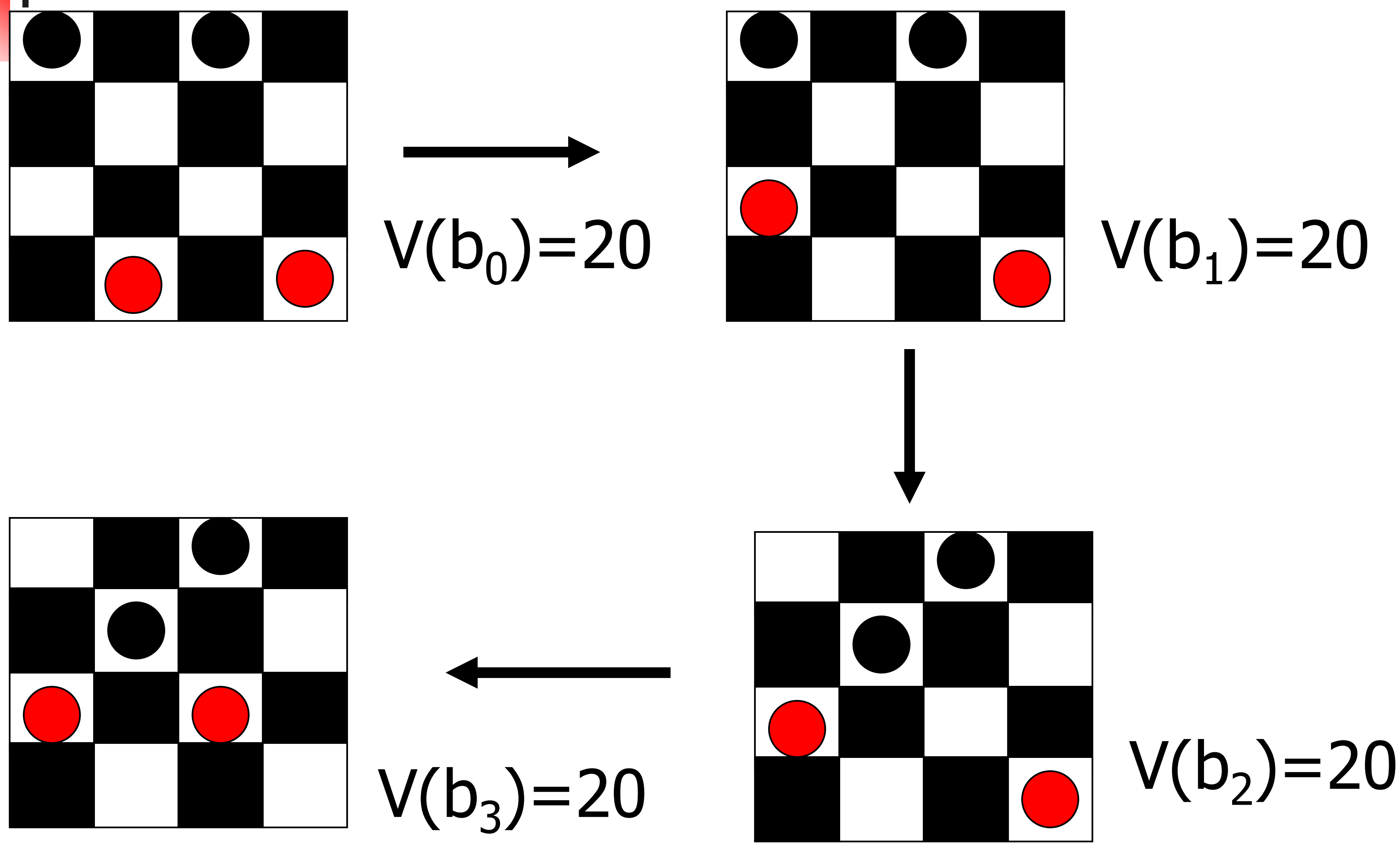
$$\omega_i \leftarrow \omega_i + \eta\ f_i\ \text{error}(b)$$

$$\boxed{\omega_0 \leftarrow \omega_0 + 0.1 * 1 * 0}$$

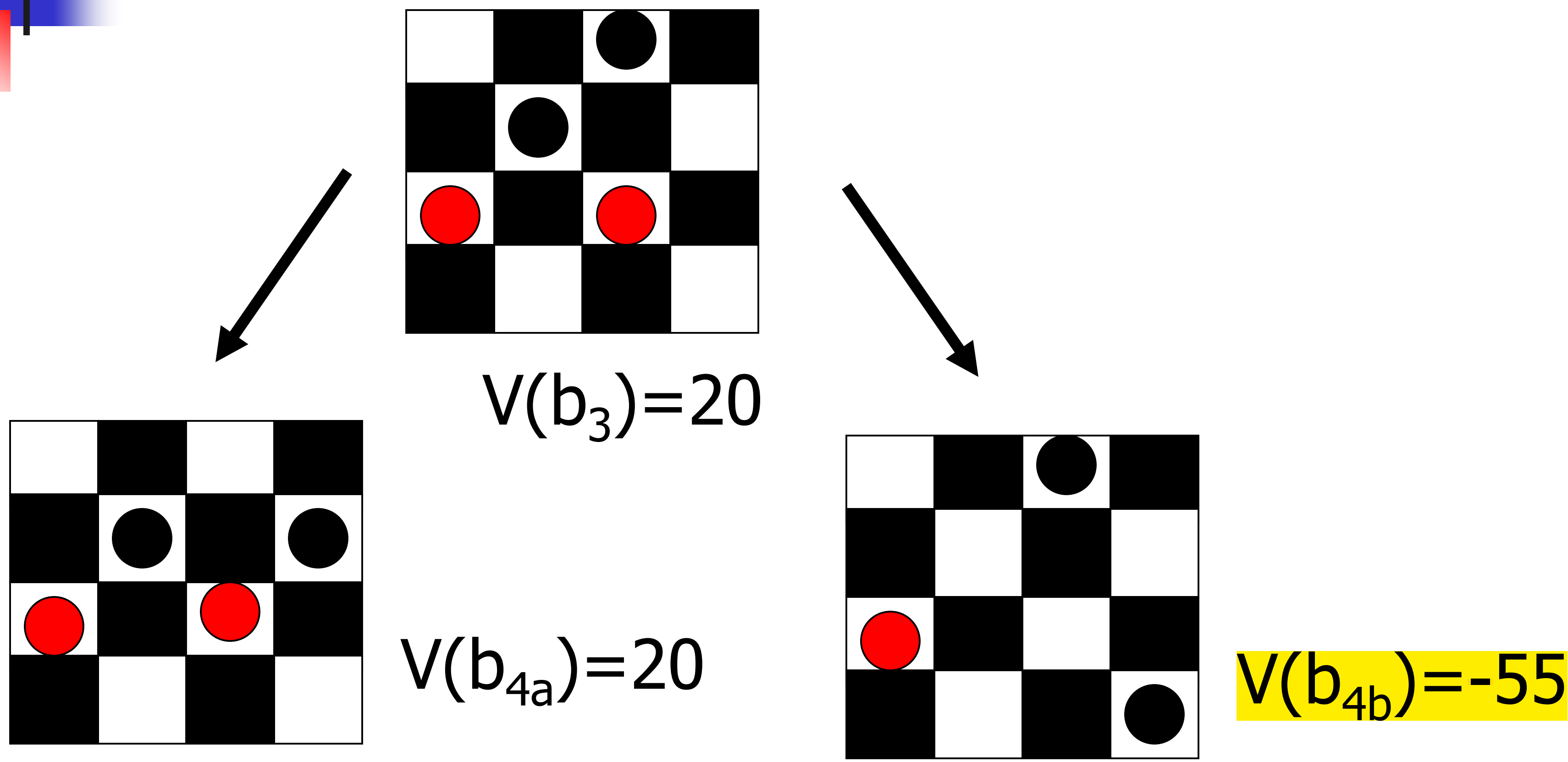
$$\omega_1 \leftarrow \omega_1 + 0.1 * 2 * 0$$

$$\omega_2 \leftarrow \omega_2 + 0.1 * 2 * 0$$

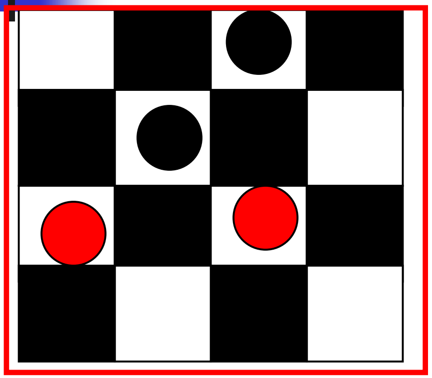
Example: 4x4 checkers



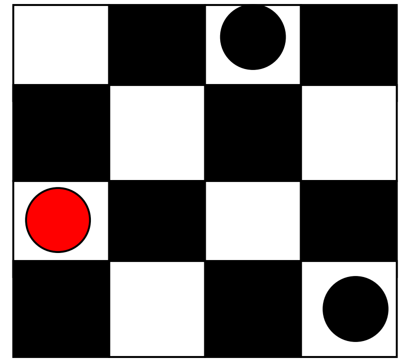
Example: 4x4 checkers



Example 4x4 checkers



$V(b_3)=20$



$V(b_4)=-55$

1. Compute $error(b_3) = V_{train}(b) – V(b_3) = V(b_4) – V(b_3) = -75$
2. For each board feature fi, update weight

$$\omega_i \leftarrow \omega_i + \eta\, f_i\, error(b) : \omega_0=-10,\ \omega_1 =75,\ \omega_2 =-60$$

$$\omega_0 \leftarrow \omega_0 - 0.1 * 1 * 7\cancel{5},\ \omega_0 = -17.5$$

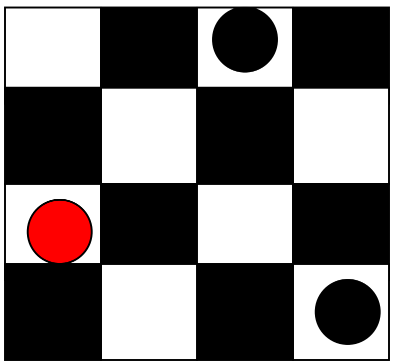
fi == 1

$$\omega_1 \leftarrow \omega_1 - 0.1 * 2 * 75,\ \omega_1 = 60$$

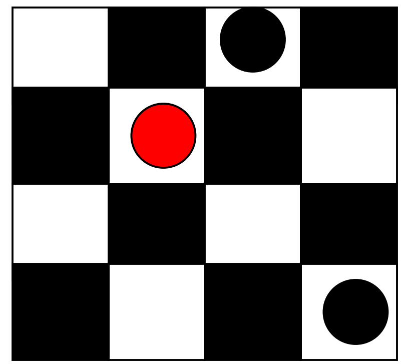
$$\omega_2 \leftarrow \omega_2 - 0.1 * 2 * 75,\ \omega_2 = -75$$

Example: 4x4 checkers

$\omega_0 = -17.5$, $\omega_1 = 60$, $\omega_2 = -75$

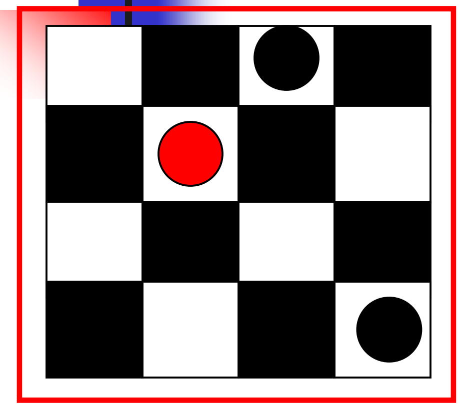


$V(b_4)$=-107.5

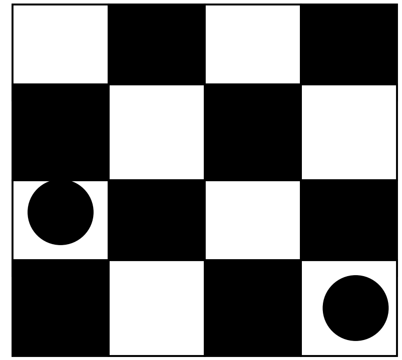


$V(b_5)$=-107.5

Example 4x4 checkers



$V(b_5)$=-107.5



$V(b_6)$=-167.5

$error(b_5) = V_{train}(b) - V(b_5) = V(b_6) - V(b_5) = -60$

ω_0=-17.5, ω_1 =60, ω_2 =-75

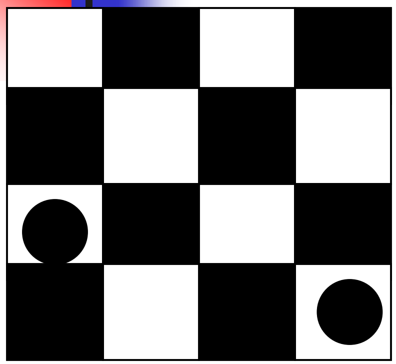
$\omega_i \leftarrow \omega_i + \eta\, f_i\, error(b)$

$\omega_0 \leftarrow \omega_0 - 0.1 * 1 * 60$, $\omega_0 = -23.5$

$\omega_1 \leftarrow \omega_1 - 0.1 * 1 * 60$, $\omega_1 = 54$

$\omega_2 \leftarrow \omega_2 - 0.1 * 2 * 60$, $\omega_2 = -87$

Example 4x4 checkers



Final board state: black won $V_f(b)$=-100

$V(b_6)$=-197.5

$$error(b_6) = V_{train}(b) - V(b_6) = V_f(b_6) - V(b_6) = 97.5$$

$$\omega_0=-23.5,\ \omega_1=54,\ \omega_2=-87$$

$$\omega_i \leftarrow \omega_i + \eta\ f_i\ error(b)$$

$$\omega_0 \leftarrow \omega_0 + 0.1 * 1 * 97.5,\ \omega_0 = -13.75$$

$$\omega_1 \leftarrow \omega_1 + 0.1 * 0 * 97.5,\ \omega_1 = 54$$

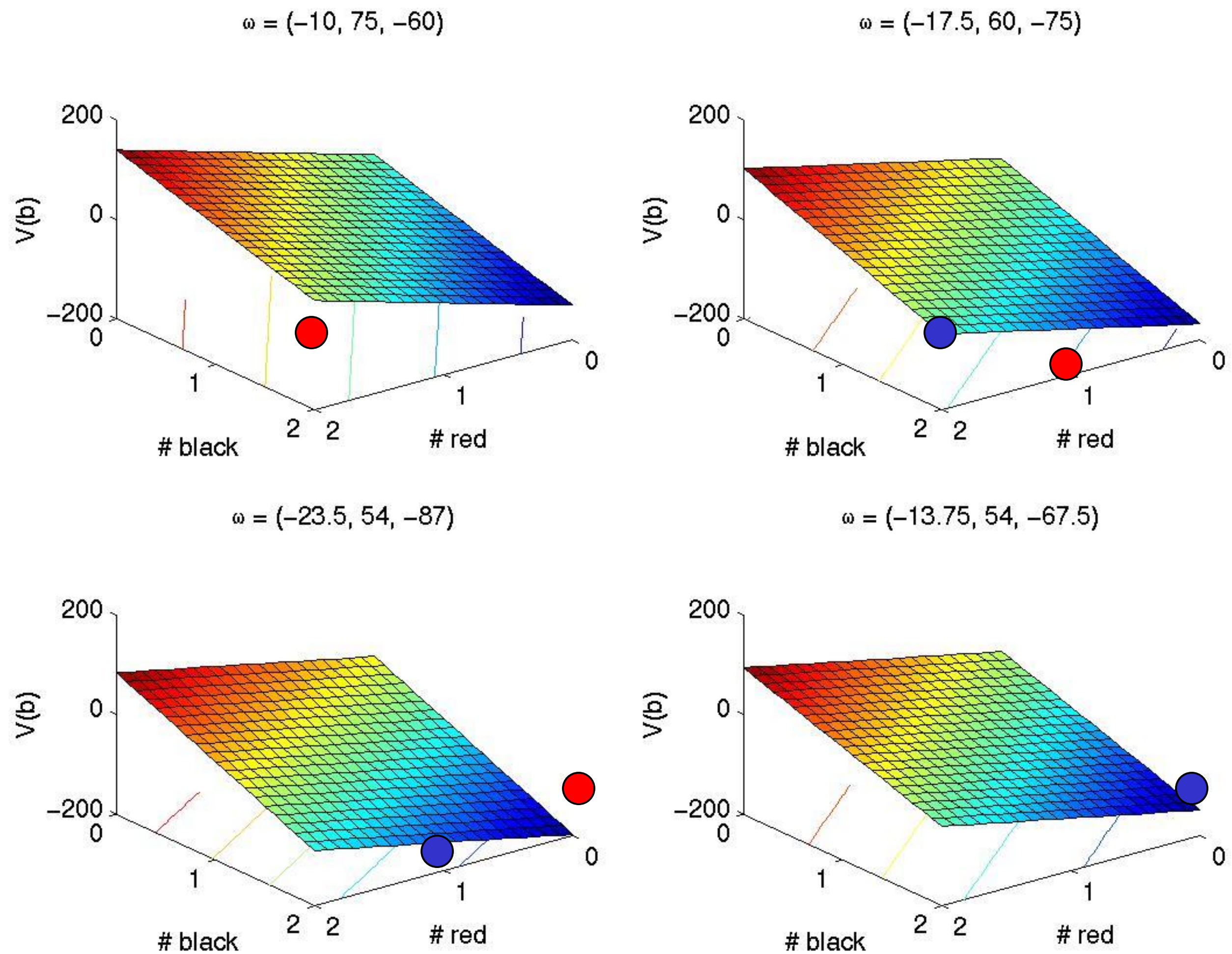
$$\omega_2 \leftarrow \omega_2 + 0.1 * 2 * 97.5,\ \omega_2 = -67.5$$

Evolution of Value Function

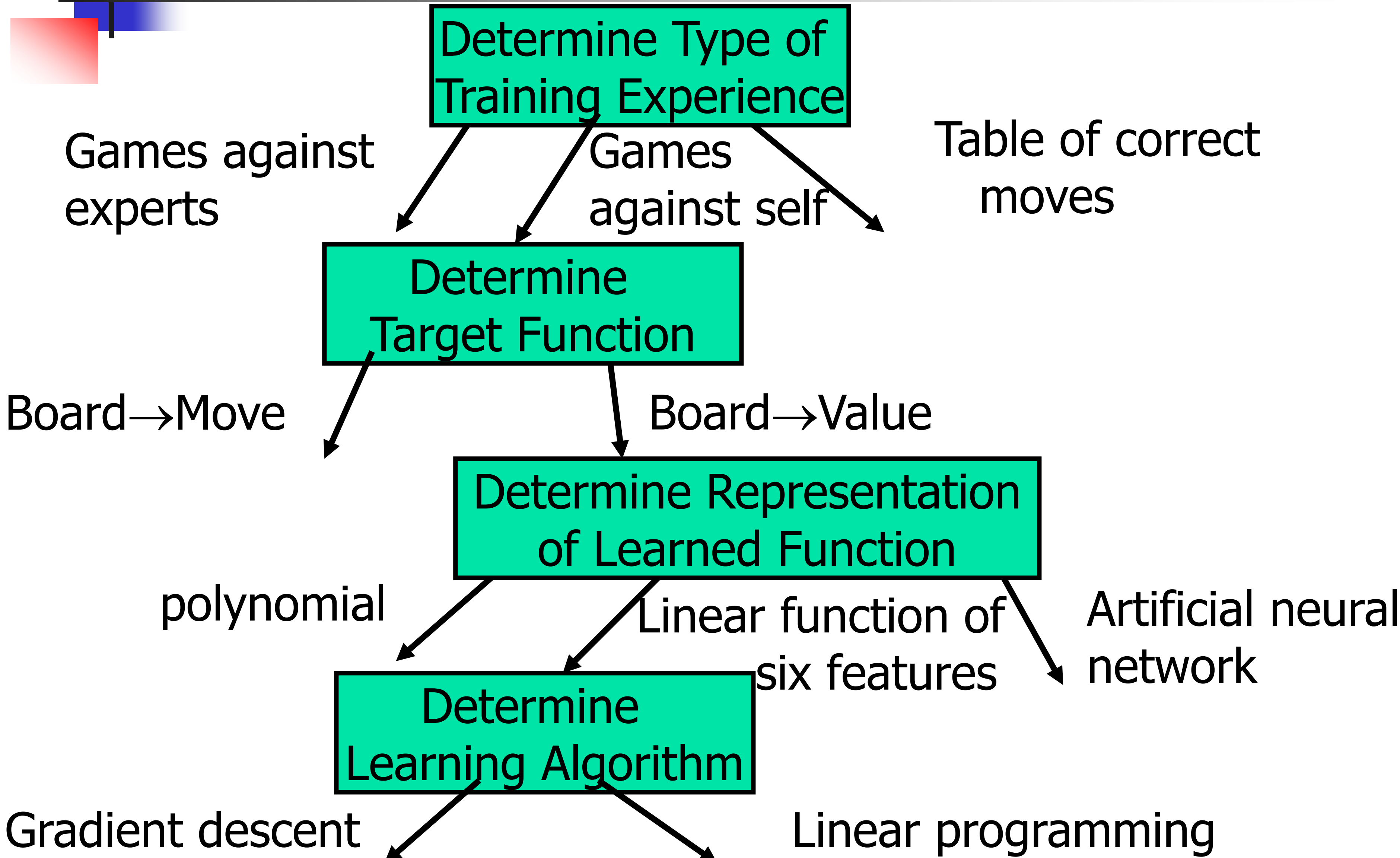
Training data:

before

after



Design Choices



Learning Problem Examples

- Credit card applications
  - Task T: Distinguish "good" applicants from "risky" applicants.
  - Performance measure P : ?
  - Experience E : ? (direct/indirect)
  - Target function : ?

Performance Measure P:

- Error based: minimize percentage of incorrectly classified customers : $P = N_{fp} + N_{fn} / N$

  N_{fp}: # false positives (rejected good customers)

  N_{fn}: # false negatives (accepted bad customers)

- Utility based: maximize expected profit of credit card business: $P = N_{cp} * U_{cp} + N_{fn} * U_{fn}$

  U_{cp} : expected utility of an accepted good customer

  U_{fn} : expected utility/loss of an accepted bad customer

Experience E:

- Direct: Decisions on credit card applications made by a human financial expert

  Training data: <customer inf., reject/accept>

- Direct: Actual customer behavior based on previously accepted customers

  Training data: <customer inf., good/bad>

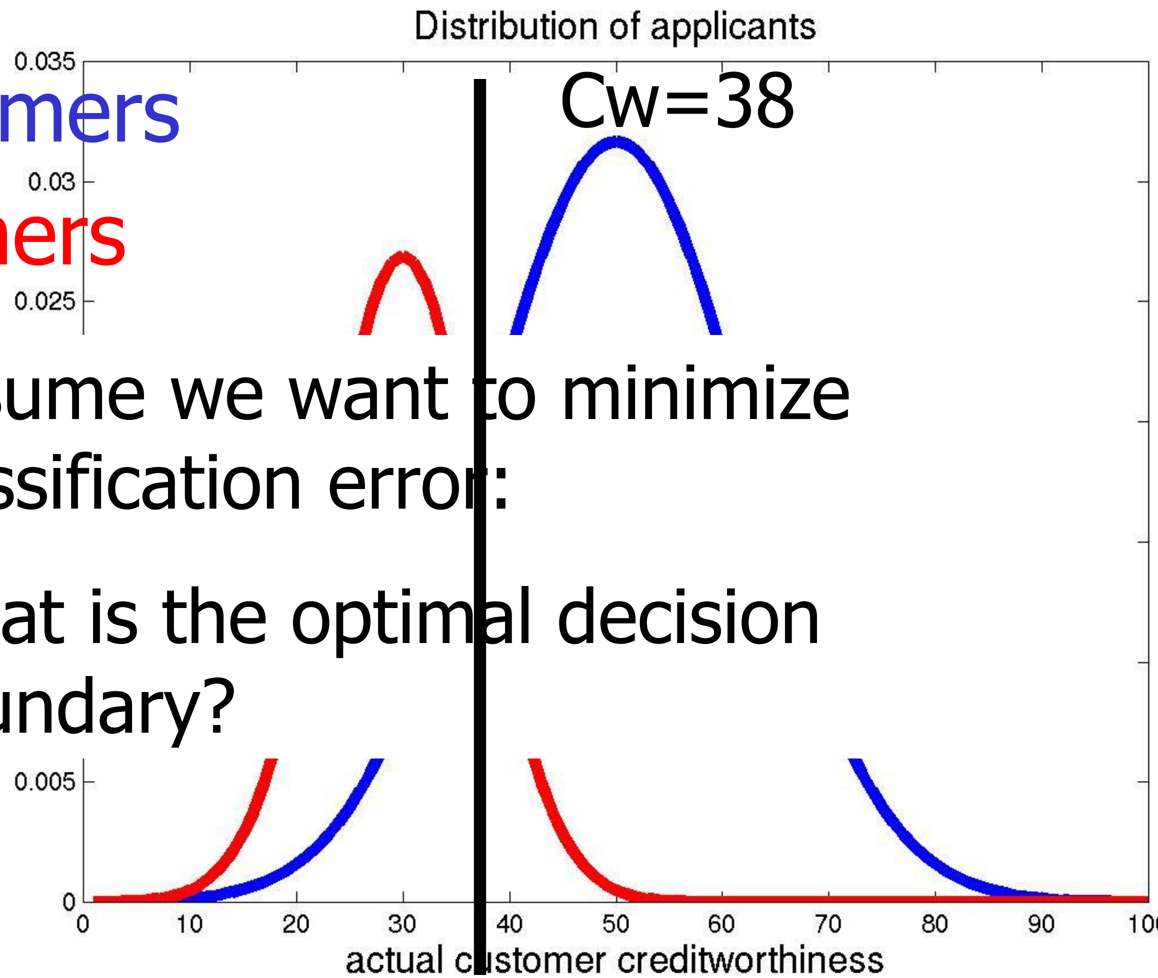
  Problem: Distribution of applicants $P_{applicant}$ is not identical with training data P_{train}

- Indirect: Evaluate a decision policy based on the profit you made over the past N years.

Distribution of Applicants

Good customers

Bad customers



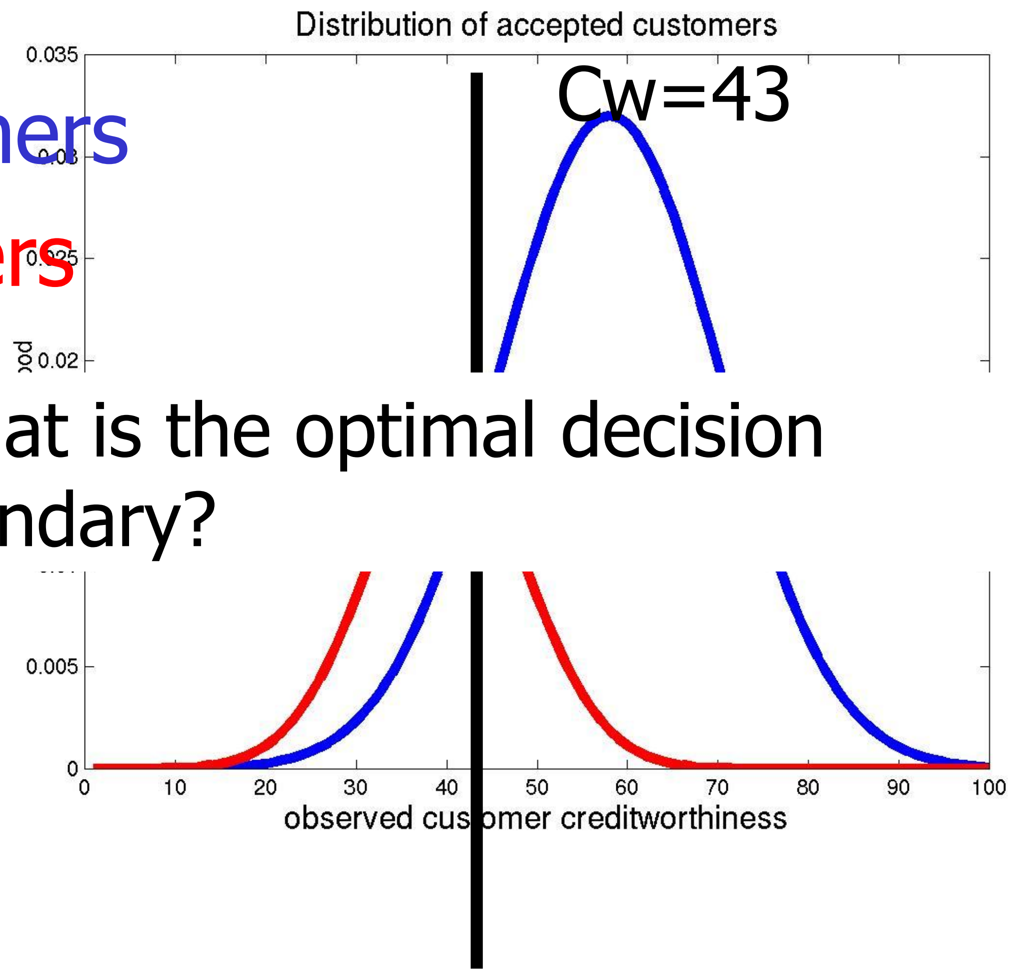
Assume we want to minimize classification error:

What is the optimal decision boundary?

Distribution of Accepted Customers

Good customers

Bad customers



What is the optimal decision boundary?

Target Function

Customer record:

income, owns house, credit history, age, employed, accept

$40000, yes, good, 38, full-time, yes

$25000, no, excellent, 25, part-time, no

$50000, no, poor, 55, unemployed, no

- T: Customer data $\rightarrow$ accept/reject
- T: Customer data $\rightarrow$ probability good customer
- T: Customer data $\rightarrow$ expected utility/profit

Learning methods

- Decision rules:
  - If income < $30.000 then reject
- Bayesian network:
  - P(good | income, credit history,….)
- Neural Network:
- Nearest Neighbor:
  - Take the same decision as for the customer in the data base that is most similar to the applicant

Learning Problem Examples

- Obstacle Avoidance Behavior of a Mobile Robot
  - Task T: Navigate robot safely through an environment.
  - Performance measure P : ?
  - Experience E : ?
  - Target function : ?

Performance Measure P:

- P: Maximize time until collision with obstacle
- P: Maximize distance travelled until collision with obstacle
- P: Minimize rotational velocity, maximize translational velocity
- P: Minimize error between control action of a human operator and robot controller in the same situation

Training Experience E:

- Direct: Monitor human operator and use her control actions as training data:
  - $E = \{ \langle perception_i, action_i \rangle \}$
- Indirect: Operate robot in the real world or in a simulation. Reward desirable states, penalize undesirable states
  - $V(b) = +1$ if $v > 0.5$ m/s
  - $V(b) = +2$ if $\omega < 10$ deg/s
  - $V(b) = -100$ if bumper state = 1

Question: Internal or external reward ?

Target Function

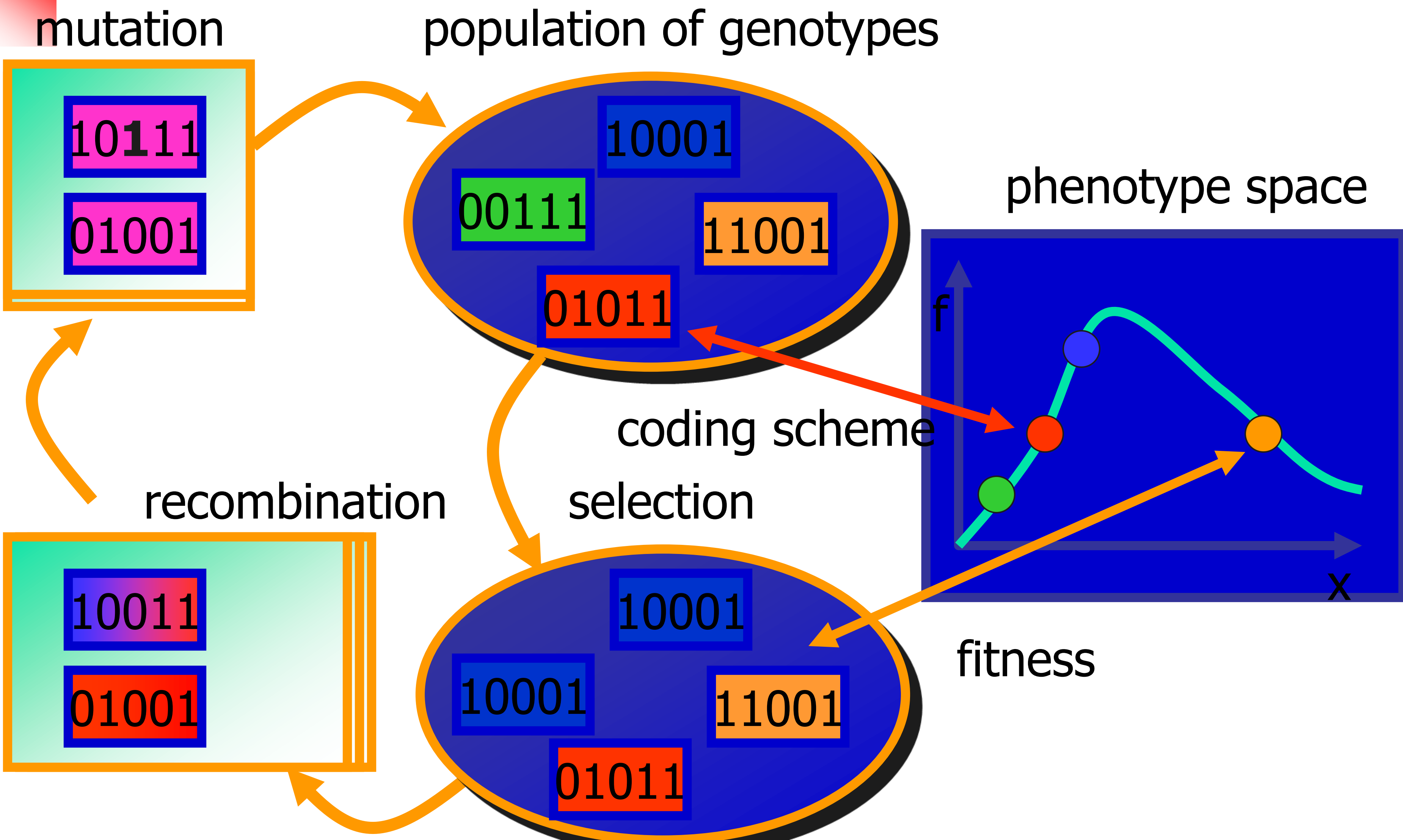
- Choose action:
  - A: perception $\rightarrow$ action

  Sonar readings: s1(t)…sn(t) $\rightarrow$ <v,ω>
- Evaluate perception/state:
  - V: s1(t)…sn(t) $\rightarrow$ V(s1(t)…sn(t))
  - Problem: states are only partially observable therefore world seems non-deterministic
  - Markov Decision Process : successor state s(t+1) is a probabilistic function of current state s(t) and action a(t)
- Evaluate state/action pairs:
  - V: s1(t)…sn(t), a(t) $\rightarrow$ V(s1(t)…sn(t),a(t))

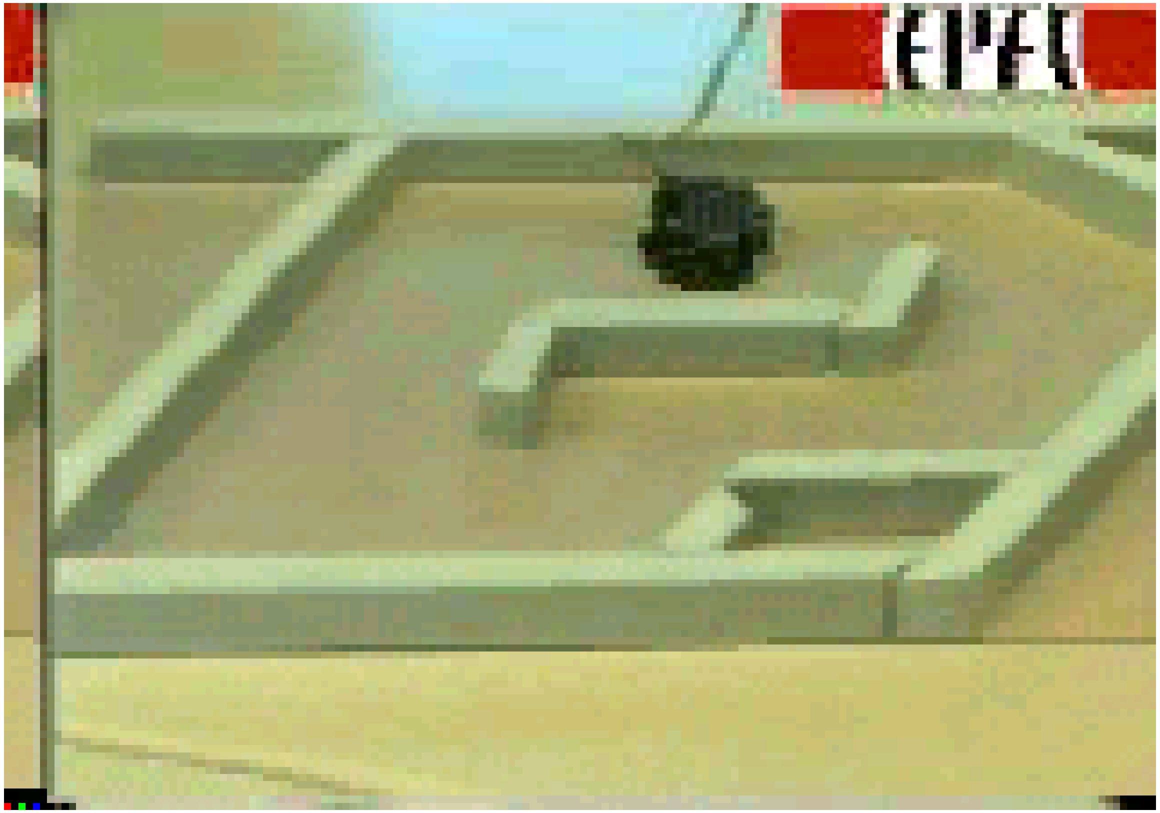
Learning Methods

- Neural Networks
  - Require direct training experience
- Reinforcement Learning
  - Indirect training experience
- Evolutionary Algorithms
  - Indirect training experience

Evolutionary Algorithms



Evolution of Simple Navigation



Issues in Machine Learning

- What algorithms can approximate functions well and when?
- How does the number of training examples influence accuracy?
- How does the complexity of hypothesis representation impact it?
- How does noisy data influence accuracy?
- What are the theoretical limits of learnability?

Machine vs. Robot Learning

Machine Learning

- Learning in vaccum
- Statistically well-behaved data
- Mostly off-line
- Informative feed-back
- Computational time not an issue
- Hardware does not matter
- Convergence proof

Robot Learning

- Embedded learning
- Data distribution not homegeneous
- Mostly on-line
- Qualitative and sparse feed-back
- Time is crucial
- Hardware is a priority
- Empirical proof

Learning in Robotics

- behavioral adaptation:
  adjust the parameters of individual behaviors according to some direct feedback signal (e.g. adaptive control)
- evolutionary adaptation:
  application of artificial evolution to robotic systems
- sensor adaptation:
  adopt the perceptual system to the environment
  (e.g. classification of different contexts, recognition)
- learning complex, deliberative behaviors:
  unsupervised learning based on sparse feedback from the environment, credit assignment problem
  (e.g. reinforcement learning)