Outline

• Rationale and motivation
• Classification and terminology
• Genetic algorithms as example of a metaheuristic
• Noise resistance and expensive optimization
• Examples
  • Standard functions
  • Control shaping
  • Hardware-software co-design
Machine-Learning for Embedded Systems: Rationale and Classification
Why Machine-Learning?

- **Complementarity to a model-based/engineering approaches**: when low-level details matter (optimization) and/or good models do not exist (design)!
- When the design/optimization space is **too big (infinite)/too computationally expensive** to be systematically searched
- **Automatic** design and optimization techniques
- **Role of engineer reduced** to specifying performance requirements and problem encoding
**Why Machine-Learning?**

- There are design and optimization techniques **robust to noise, nonlinearities, discontinuities**
- Individual **real-time adaptation** to new environmental conditions; i.e. increased individual flexibility when environmental conditions are not known/cannot predicted a priori
- Search space: **parameters and/or rules**
ML Techniques: Classification Axis 1

- **Supervised learning**: off-line, a teacher is available

- **Unsupervised learning**: off-line, teacher not available

- **Reinforcement (or evaluative) learning**: on-line, no pre-established training and evaluation data sets
Supervised Learning

- Off-line
- Training and test data are separated, a teacher is available
- Typical scenario: a set of input-output examples is provided to the system, performance error given by difference between system output and true/teacher-defined output, error fed to the system using optimization algorithm so that performance is increased over trial
- The generality of the system after training is tested on examples not previously presented to the system (i.e. a “test set” exclusive from the “training set”)
Unsupervised Learning

- Off-line
- No teacher available, no distinction between training and test data sets
- Goal: structure extraction from the data set
- Examples: data clustering, Principal Component Analysis (PCA) and Independent Component Analysis (ICA)
Reinforcement (or Evaluative) Learning

- On-line
- No pre-established training or test data sets
- The system judges its performance according to a given metric (e.g., fitness function, objective function, performance, reinforcement) to be optimized
- The metric does not refer to any specific input-to-output mapping
- The system tries out possible design solutions, does mistakes, and tries to learn from its mistakes
ML Techniques: Classification Axis 2

- **In simulation**: reproduces the real scenario in simulation and applies there machine-learning techniques; the learned solutions are then downloaded onto real hardware when certain criteria are met.

- **Hybrid**: most of the time in simulation (e.g. 90%), last period (e.g. 10%) of the learning process on real hardware.

- **Hardware-in-the-loop**: from the beginning on real hardware (no simulation). Depending on the algorithm more or less rapid.
ML Techniques: Classification Axis 3

ML algorithms require sometimes fairly important computational resources (in particular for multi-agent optimization algorithms), therefore a further classification is:

- **On-board**: machine-learning algorithm run on the system to be learned (no external unit)

- **Off-board**: the machine-learning algorithm runs off-board and the system to be learned just serves as embodied implementation of a candidate solution
Selected ML Techniques Robust to Noisy Performance and Discontinuous Search Space

• Evolutionary computation
  – Genetic Algorithms (GA) ➔ This course
  – Genetic Programming (GP)
  – Evolutionary Strategies (ES)
  – Particle Swarm Optimization (PSO) ➔ DIS course

• Learning
  – In-Line Adaptive Learning ➔ DIS course
  – Reinforcement Learning (RL) ➔ DIS course
Genetic Algorithms
Genetic Algorithms Inspiration

• In natural evolution, organisms adapt to their environments – better able to survive over time

• Aspects of evolution:
  – Survival of the fittest
  – Genetic combination in reproduction
  – Mutation

• Genetic Algorithms use evolutionary techniques to achieve parameter optimization and solution design
GA: Terminology

- **Population**: set of m candidate solutions (e.g. m = 100); each candidate solution can also be considered as a genetic individual endowed with a single chromosome which in turn consists of multiple genes.

- **Generation**: new population after genetic operators have been applied (n = # generations e.g. 50, 100, 1000).

- **Fitness function**: measurement of the efficacy of each candidate solution

- **Evaluation span**: evaluation period of each candidate solution during a given generation. The time cost of the evaluation span differs greatly from scenario to scenario: it can be extremely cheap (e.g., simply computing the fitness function in a benchmark function) or involve an experimental period (e.g., evaluating the performance of a given control parameter set on a robot)

- **Life span**: number of generations a candidate solution survives

- **Population manager**: applies genetic operators to generate the candidate solutions of the new generation from the current one
Evolutionary Loop: Several Generations

Ex. of end criteria:
• # of generations
• best solution performance
• …
Generation Loop

Population replenishing → Crossover and mutation → Selection → Fitness measurement and encoding (phenotype → genotype) → Decoding (genotype → phenotype) → Evaluation of Individual Candidate Solutions → Population Manager
GA: Encoding & Decoding

phenotype → genotype → phenotype
encoding (chromosome) decoding

- phenotype: usually represented by the whole system which can be evaluated; the whole system or a specific part of it (problem formulation done by the engineer) is represented by a vector of dimension D; vector components are usually real numbers in a bounded range.

- genotype: chromosome = string of genotypical segments, i.e. genes, or mathematically speaking, again a vector of real or binary numbers; vector dimension varies according to coding schema ($\geq D$); the algorithm search in this hyperspace.

\[
\begin{array}{cccccc}
  G_1 & G_2 & G_3 & G_4 & \ldots & G_n \\
\end{array}
\]

\(G_i = \text{gene} = \text{binary or real number}\)

**Encoding:** real-to-real or real-to-binary via Gray code (minimization of nonlinear jumping between phenotype and genotype)

**Decoding:** inverted operation

**Rem:**

- **Artificial evolution:** usually one-to-one mapping between phenotypic and genotypic space.

- **Natural evolution:** 1 gene codes for several functions, 1 function coded by several genes.
GA: Basic Operators

- **Selection**: *roulette wheel* (selection probability determined by normalized fitness), *ranked selection* (selection probability determined by fitness order), *elitist selection* (highest fitness individuals always selected)

- **Crossover**: 1 point, 2 points (e.g. $p_{\text{crossover}} = 0.2$)

- **Mutation** (e.g. $p_{\text{mutation}} = 0.05$)

Note: examples for fixed-length chromosomes!
GA: Discrete vs Continuous

• For default GA, all parameters discrete (e.g., binary bits, choice index)
• Common adaptation for continuous optimization:
  – Parameters are real values
  – Mutation: apply randomized adjustment to gene value (i.e. $G_i' = G_i + m$) instead of replacing value
• Selection of adjustment range affects optimization progress
Learning to Avoid Obstacles by Shaping a Neural Network Controller using Genetic Algorithms
Evolving a Neural Controller

The output of neuron $N_i$ with sigmoid transfer function $f(x)$ is given by:

$$O_i = f(x_i)$$

where

$$f(x) = \frac{2}{1 + e^{-x}} - 1$$

The input $x_i$ to neuron $N_i$ is given by:

$$x_i = \sum_{j=1}^{m} w_{ij} I_j + I_0$$

Note: In our case we evolve synaptic weights but Hebbian rules for dynamic change of the weights, transfer function parameters, … can also be evolved (see Floreano’s course)
Evolving Obstacle Avoidance
(Floreano and Mondada 1996)

Defining performance (fitness function):

\[ \Phi = V (1 - \sqrt{\Delta V})(1 - i) \]

- \( V = \) mean speed of wheels, \( 0 \leq V \leq 1 \)
- \( \Delta v = \) absolute algebraic difference between wheel speeds, \( 0 \leq \Delta v \leq 1 \)
- \( i = \) activation value of the sensor with the highest activity, \( 0 \leq i \leq 1 \)

**Note:** Fitness accumulated during evaluation span, normalized over number of control loops (actions).
Note:
Controller architecture can be of any type but worth using GA if the number of parameters to be tuned is important.
Evolved Obstacle Avoidance Behavior

Note: Direction of motion (forwards vs. backwards) NOT encoded in the fitness function: GA automatically discovers asymmetry in the sensory system configuration (6 proximity sensors in the front and 2 in the back)

Generation 100, on-line, off-board (PC-hosted) evolution
Evolving Obstacle Avoidance

Evolved path

Fitness evolution
Expensive Optimization and Noise Resistance
Expensive Optimization Problems

Two fundamental reasons making embedded system design and optimization expensive in terms of time:

1. **Time for evaluation** of candidate solutions (e.g., tens of seconds) >> time for application of metaheuristic operators (e.g., ms)

2. **Noisy performance evaluation** disrupts learning process and require multiple evaluations for actual performance
   - Multiple evaluations at the same point in the search space yield different results
   - Noise causes decreased convergence speed and residual error
Algorithmic Principles for Dealing with Expensive Optimization Problems

- Better information about candidate solution can be obtained by combining multiple noisy evaluations
- We could evaluate systematically each candidate solution for a fixed number of times → not efficient from a computational perspective
- We want to dedicate more computational time to evaluate promising solutions and eliminate as quickly as possible the “lucky” ones → each candidate solution might have been evaluated a different number of times when compared
- In GA good and robust candidate solutions survive over generations
- Use dedicated functions for aggregating multiple evaluations: e.g., minimum and average (perhaps combined with a statistical test)
Noise-Resistant GA

Dynamic population management during evaluation/selection: competition between offspring (new candidate solutions) and re-evaluated parents (old candidate solutions)
Testing Noise-Resistant GA on Benchmarks

• Benchmark 1: generalized Rosenbrock function
  – 30 real parameters
  – Minimize objective function
  – Expensive only because of noise

• Benchmark 2: obstacle avoidance on a robot
  – 22 real parameters
  – Maximize objective function
  – Expensive because of noise and evaluation time
Benchmark 1: Gaussian Additive Noise on Generalized Rosenbrock

\[ f'_j(x) = N(0, \sigma^2) + f_j(x) \]

**Fair test:** same number of evaluations candidate solutions for all algorithms (i.e. n generations/iterations of standard versions compared with n/2 of the noise-resistant ones)
Benchmark 2: Obstacle Avoidance on a Mobile Robot

- Discrete-time, single-layer, artificial neural network controller
- Learning: neural weights and biases (22 real parameters)
- Fitness function (Floreano and Mondada 1996) rewards speed, straight movement, avoiding obstacles:

\[ F = V \cdot (1 - \sqrt{\Delta v}) \cdot (1 - i) \]

\[ 0 \leq V \leq 1, \ 0 \leq \Delta v \leq 1, \ 0 \leq i \leq 1 \]

- \( V \) = average wheel speed, \( \Delta v \) = difference between wheel speeds, \( i \) = value of most active proximity sensor
Extended-Time Robotic Learning

- Compare GA and Noise-Resistant GA (PSO solution not considered in this course)
- Population size 100, 100 iterations, evaluation span 300 seconds → 34.7 days
- Similar performance for all algorithms
- Module-based simulation (Webots)
Limited-Time Learning Trade-Offs

- Total learning time = 8.3 hours (1/100 of previous learning time)
- Trade-offs: population size, number of iterations, evaluation span
- Module-based simulation (Webots)

Especially good with small populations

Varying population size vs. number of iterations
Hybrid Learning with Real Robots

- Move from simulation (module-based, Webots) to real robots after 90% learning (even faster evolution)
- Compromise between time and accuracy
- Noise-resistance helps manage transition
Not only Obstacle Avoidance: Evolving More Complex Behaviors
Evolving Homing Behavior
(Floreano and Mondada 1996)

Set-up

Robot’s sensors
Evolving Homing Behavior

- **Fitness function:**

\[ \Phi = V(1 - i) \]

- \( V \) = mean speed of wheels, \( 0 \leq V \leq 1 \)
- \( i \) = activation value of the sensor with the highest activity, \( 0 \leq i \leq 1 \)

- Fitness accumulated during life span, normalized over maximal number (150) of control loops (actions).
- No explicit expression of battery level/duration in the fitness function (implicit).
- Chromosome length: 102 parameters (real-to-real encoding).
- Generations: 240, 10 days embedded evolution on Khepera.
Evolving Homing Behavior

Fitness evolution

Evolution of # control loops per evaluation span

Battery recharging vs. motion patterns

Battery energy
Left wheel activation
Right wheel activation

Reach the nest -> battery recharging -> turn on spot -> out of the nest
Evolved Homing Behavior
Evolving Homing Behavior

Activation of the fourth neuron in the hidden layer

Firing is a function of:

- battery level
- orientation (in comparison to light source)
- position in the arena (distance from light source)
Not only Control Shaping: Off-line Automatic Hardware-Software Co-Design and Optimization
Moving Beyond Controller-Only Evolution

• Evidence: Nature evolve HW and SW at the same time …

• Faithful realistic simulators enable to explore design solution which encompasses evolution of HW features (body shape, number of sensors, placement of sensors, etc.) or co-evolution of both control and HW features

• GA are powerful enough for this job and the methodology remain the same; only the problem formulation changes
Multi-Vehicle Scenario

Webots 2004

Webots 2009
Evolving Sensory Configuration

- **Number** $n$ (variable chromosome length!)
- **Type**
  - Range $\rho$
  - Cone of view $\delta$
  - Cost $f(\rho, \delta)$
- **Placement**
  - Position $\varphi$
  - Orientation $\theta$

[Zhang et al., RED 2008]
Fitness Function

\[ \text{Fitness}(\omega, s) = \left( \frac{\mu_{\text{cost}}^s + \omega \cdot \mu_{\text{coverage}}^s}{1 + \omega} \right)^{\frac{1}{s}} \]

\[ \omega = \frac{\omega_{\text{coverage}}}{\omega_{\text{cost}}} \]

\[ \text{Coverage} = \sum_{i=1}^{V} k_i \text{PDF}(\alpha_i, r_i) \]

\[ \text{Cost} = \sum_{i=1}^{n} \text{cost}_i \]

\[ \text{cost}_i = f(\rho_i, \delta_i) \]

\( \mu_{xx} \): fuzzy preference on factor \( xx \)
- 0: totally unacceptable;
- 1: completely acceptable

\( \omega_{xx} \): weight on factor \( xx \)

\( s \): degree of compensation

\( V \): actual number of vehicles
- been in the detection region during an evaluation span

\( k_i \): 1 if the \( i^{th} \) car at distance \( r_i \)
- and approaching angle \( \alpha_i \)
- detected; 0 if not detected

\( n \): number of sensors used

\( \rho_i \): \( i^{th} \) sensor’s range

\( \delta_i \): \( i^{th} \) sensor’s cone of view

\( \text{PDF} \): probability density function

\( \sum \): summation symbol

\( f \): function symbol
Sample Results

- Fitness evolution process and the final best design

$s = 0, \omega = 3/17$

Coverage = 53%, Total_cost = 4.6
Sample Results

- Fitness evolution process and the final best design

\[ s = -\infty, \omega \text{ arbitrary} \]

\[ \text{Coverage} = 82\%, \quad \text{Total\_cost} = 7.7 \]
Sample Results

- Fitness evolution process and the final best design

$s = 0, \omega = 4$

Coverage = 98\%, Total\_cost = 11.9
Sample Results

- Evolved approximate Pareto frontier for the design trade-offs present in the case study

[Zhang et al., RED 2008]
Evolving Control and Robot Morphology (Lipson and Pollack, 2000)

http://www.mae.cornell.edu/ccsl/research/golem/index.html

- Arbitrary recurrent ANN
- Passive and active (linear actuators) links
- **Fitness function**: net distance traveled by the centre of mass in a fixed duration

Example of evolutionary sequence:
Examples of Evolved Machines

**Problem:** simulator not enough realistic (performance higher in simulation because of not good enough simulated friction; e.g., for the arrow configuration 59.6 cm vs. 22.5 cm)
Conclusion
Take Home Messages

• Key classification in machine-learning is supervised, unsupervised, and reinforcement (evaluative) learning

• Reinforcement/evaluative techniques are key for on-line robotic learning

• A robust multi-agent metaheuristic is GA and can be successfully combined with ANN

• GA can be used to shape the behavior by tuning ANN synaptic weights

• Computationally efficient, noise-resistant algorithms can be obtained with a simple aggregation criterion in the main evolutionary loop
Additional Literature – Week 11

Books
- Nolfi S. and Floreano D., “Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines”. MIT Press, 2004

PhD Thesis

Papers