Reinforcement Learning

Michèle Sebag ; TP : Diviyan Kalainathan, Laurent Cétinsoy
TAO, CNRS – INRIA – Université Paris-Sud

Jan. 24th, 2018

Where we are

MDP Main Building block

General settings

	Model-based	Model-free
Finite	Dynamic Programming	Discrete RL
Infinite	(optimal control)	**Continuous RL**

Last course: Function approximation
This course: Direct policy search; Evolutionary robotics

Position of the problem

Notations

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Transition model $p(s, a, s') \mapsto [0, 1]$
- Reward $r(s)$ bounded

Mainstream RL: based on values

$$V^* : \mathcal{S} \mapsto \mathbb{R} \qquad \pi^*(s) = \underset{a \in \mathcal{A}}{arg\ opt} \left(\sum_{s'} p(s, a, s') V^*(s') \right)$$

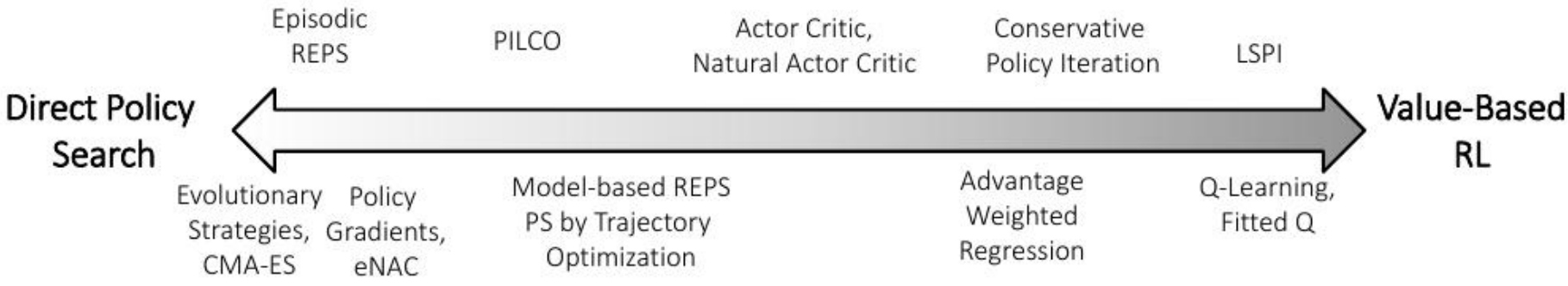
$$Q^* : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R} \qquad \pi^*(s) = \underset{a \in \mathcal{A}}{arg\ opt} \left(Q^*(s, a) \right)$$

What we want

$$\pi : \mathcal{S} \mapsto \mathcal{A}$$

Aren't we learning something more complex than needed ?...
$\Rightarrow$ Let us consider Direct policy search

From RL to Direct Policy Search



Direct policy search: define

- Search space (representation of solutions)
- Optimization criterion
- Optimization algorithm

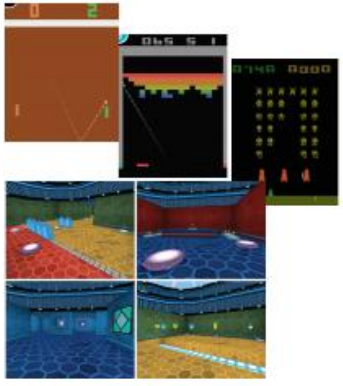
Examples

Kohl and Stone, 2004

Ng et al, 2004

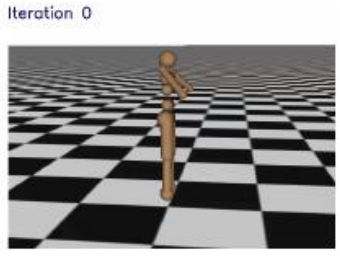
Tedrake et al, 2005

Kober and Peters, 2009

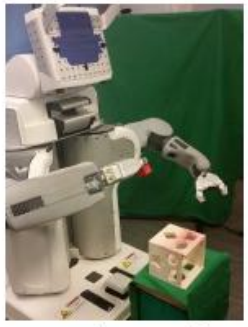


Mnih et al, 2015
(A3C)

Silver et al, 2014
(DPG)
Lillicrap et al, 2015
(DDPG)



Schulman et al,
2016 (TRPO + GAE)



Levine*, Finn*, et
al, 2016
(GPS)

Silver*, Huang*, et
al, 2016
(AlphaGo**)

Representation

1.Explicit representation $\equiv$ Policy space

π is represented as a function from $\mathcal{S}$ onto $\mathcal{A}$

- Non-parametric representation, e.g. decision tree or random forest
- Parametric representation. Given a function space, π is defined by a vector of parameters θ.

$$\pi_\theta = \begin{cases} \text{Linear function on } \mathcal{S} \\ \text{Radius-based function on } \mathcal{S} \\ \text{(deep) Neural net} \end{cases}$$

E.g. in the linear function case, given $s \in \mathcal{S} = \mathbb{R}^d$ and θ in $\mathbb{R}^d$,

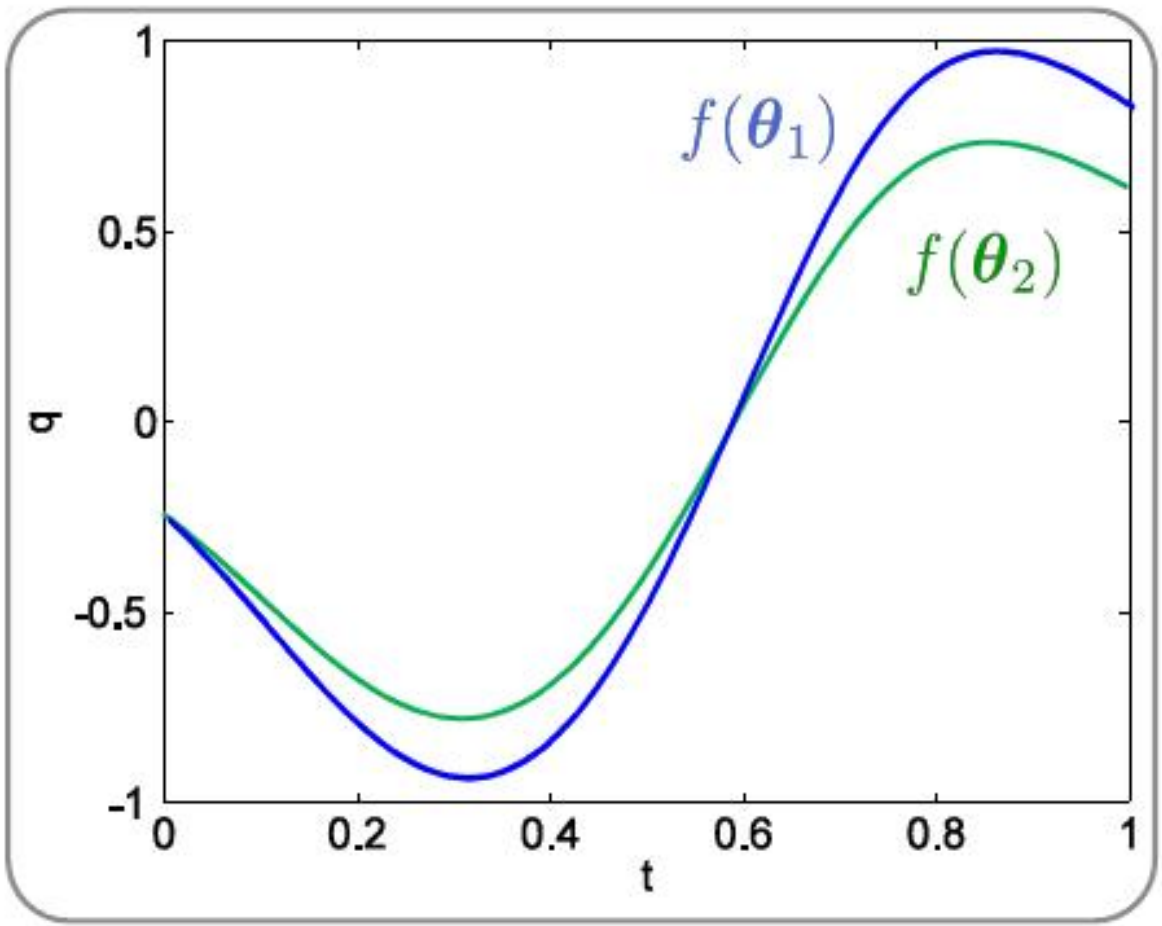
$$\pi_\theta(s) = \langle s, \theta \rangle$$

Representation

2. Implicit representation: for example Trajectory generators

$\pi(s)$ is obtained by solving an auxiliary problem. For instance,

- Define desired trajectories — Dynamic movement primitives
- Trajectory $\tau = f(\theta)$
- Action = getting back to the trajectory given the current state s



Direct policy search in RL

Two approaches

- Model-free approaches
- Model-based approaches

History

- Model-free approaches were the first ones; they work well but i) require many examples; ii) these examples must be used in a smart way.
- Model-based approaches are more recent. They proceed by i) modelling the MDP from examples (this learning step has to be smart); ii) using the model as if it were a simulator.
  Important points: the model must give a prediction **and** a confidence interval (will be very important for the exploration).

DPS: The model-free approach

DPS: The model-based approach

- Gaussian processes

Evolutionary robotics

- Reminder
- Evolution of morphology

Others

The model-free approach

Algorithm

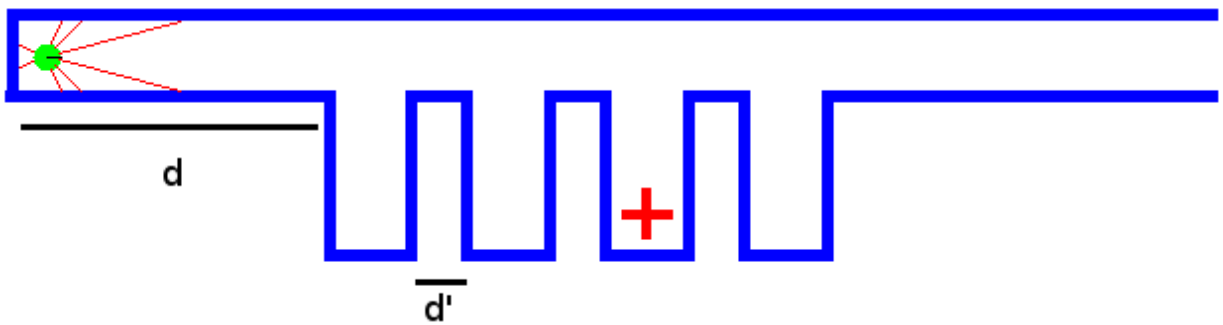
1. Explore: Generate trajectories $\tau_i = (s_{i,t}, a_{i,t})_{t=1}^{T}$ after π_{θ_k}
2. Evaluate:
   - Compute quality of trajectories — Episode-based
   - Compute quality of (state-action) pairs — Step-based
3. Update: compute θ_{k+1}

Two modes

- Episode-based
  - learn a distribution $\mathcal{D}_k$ over Θ
  - draw θ after $\mathcal{D}_k$, generate trajectory, measure its quality
  - bias $\mathcal{D}_k$ toward the high quality regions in Θ space
- Step-based
  - draw a_t from $\pi(s_t, \theta_k)$
  - measure $q_\theta(s, a)$ from the cumulative reward gathered after having visited (s, a)

Model-free Episode-based DPS. PROS

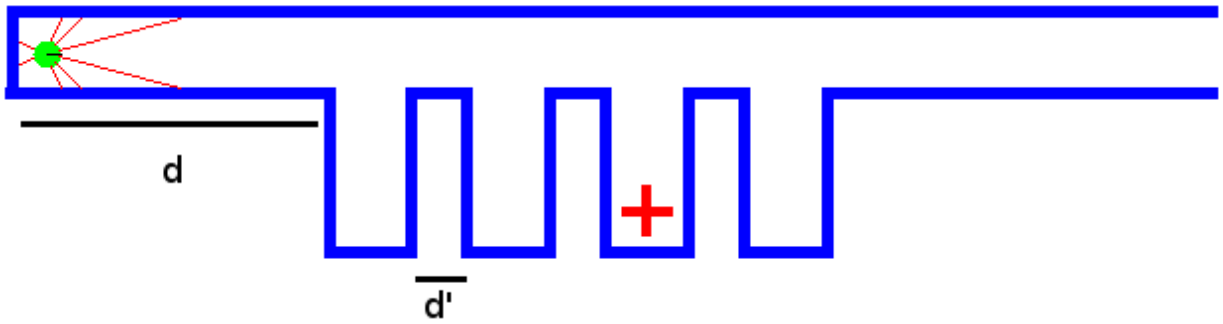
Getting rid of Markovian assumption



-

Model-free Episode-based DPS. PROS

Getting rid of Markovian assumption



- Rover on Mars: take a picture of region 1, region 2, ...

PROS, 2

Hopes of scalability

- With respect to continuous state space
- No divergence even under function approximation

Tackling more ambitious goals also see Evolutionary RL

- Partial observability does not hurt convergence (though increases computational cost)
- Optimize controller (software) and also morphology of the robot (hardware);
- Possibly consider co-operation of several robots...

Model-free Episode-based DPS. CONS

Lost the global optimum properties

- Not a well-posed optimization problem in general
- Lost the Bellman equation $\Rightarrow$ larger variance of solutions

A noisy optimization problem

- Policy $\pi \rightarrow$ a distribution over the trajectories (depending on starting point, on noise in the environment, sensors, actuators...)
- $V(\theta) =_{def} \mathbb{E}\left[\sum_t \gamma^t r_{t+1} | \theta\right]$ or

$$V(\theta) =_{def} \mathbb{E}_\theta \left[J(\text{ trajectory })\right]$$

- In practice

$$V(\theta) \approx \frac{1}{K} \sum_{i=1}^{K} J(\text{ trajectory }_i)$$

How many trajectories are needed ?

Requires tons of examples

CONS, 2

The in-situ *vs* in-silico dilemma

- In-situ: launch the robot in the real-life and observe what happens
- In-silico: use a simulator
  - But is the simulator realistic ???

The exploration *vs* exploitation dilemma

- For generating the new trajectories
- For updating the current solution θ

$$\theta_{t+1} = \theta_t - \alpha_t \nabla V(\theta)$$

Very sensitive to the learning rate α_t.

The model-free approach, how

An optimization objective

An optimization mechanism

- Gradient-based optimization
- Define basis functions ϕ_i, learn α_i
- Use black-box optimization

Cumulative value, gradient

The cumulative discounted value

$$V(s_0) = r(s) + \sum_{t=1} \gamma^t r(s_t)$$

with s_{t+1} next state after s_t for policy π_θ

The gradient

$$\frac{\partial V(s_0, \theta)}{\partial \theta} \approx \frac{V(s_0, \theta + \epsilon) - V(s_0, \theta - \epsilon)}{2\epsilon}$$

- Model $p(s_{t+1}|s_t, a_t, \theta)$ not required but useful
- Laarge variance ! many samples needed.

A trick

- Using a simulator: Fix the random seed and reset
- No variance of $V(s_0, \theta)$, much smaller variance of its gradient

Average value, gradient

No discount: long term average reward

$$V(s) = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}\left[\sum_t r(s_t) | s_0 = s\right]$$

Assumption: ergodic Markov chain

(After a while, the initial state does not matter).

- $V(s)$ does not depend on s
- One can estimate the percentage of time spent in state s

$$q(\theta, s) = Pr_\theta(S = s)$$

Yields another value to optimize

$$V(\theta) = \mathbb{E}_\theta[r(S)] = \sum_s r(s) q(\theta, s)$$

Model-free Direct Policy Search

Algorithm

1. $V(\theta) = \mathbb{E}_\theta[r(S)] = \sum_s r(s)q(\theta, s)$
2. Compute or estimate the gradient $\nabla V(\theta)$
3. $\theta_{t+1} = \theta_t + \alpha_t \nabla V(\theta)$

Computing the derivative

$$\nabla V = \nabla \left(\sum_s r(s)q(\theta, s) \right) = \sum_s r(s) \nabla q(\theta, s)$$

$$= \mathbb{E}_{S,\theta} \left[r(S) \frac{\nabla q(\theta, S)}{q(\theta, S)} \right]$$

$$= \mathbb{E}_{S,\theta} \left[r(S) \nabla log\, q(\theta, S) \right]$$

Unbiased estimate of the gradient ($\widehat{\text{integral}}$ = empirical sum)

$$\hat{\nabla} V = \frac{1}{N} \sum_i r(s_i) \frac{\nabla q(\theta, s_i)}{q(\theta, s_i)}$$

The Success Matching Principle

$$\pi_{new}(a|s) \propto \text{ Success } (s, a, \theta).\pi_{old}(a|s)$$

Different computations of "Success"

- $\theta \sim \mathcal{D}_k$ generates trajectory, evaluation $V(\theta)$
- Transform evaluation into (non-negative) probability w_k
- Find mixture policy π_{k+1}

$$p(a|s) \propto \sum w_k p(a|s, \theta_k)$$

- Find θ_{k+1} accounting for π_{k+1}
- Update $\mathcal{D}_k$, iterate

Computing the weights

$$w_k = exp\left(\beta(V(\theta) - minV(\theta))\right)$$

β: temperature of optimization simulated annealing

Example

$$= exp\left(10\frac{V(\theta) - minV(\theta)}{maxV(\theta) - minV(\theta)}\right)$$

Model-free Direct Policy Search, summary

Algorithm

- Define the criterion to be optimized (cumulative value, average value)
- Define the search space (Θ: parametric representation of π)
- Optimize it: $\theta_k \rightarrow \theta_k + 1$
  - Using gradient approaches
  - Updating a distribution $\mathcal{D}_k$ on Θ
  - In the step-based mode or success matching case:
    find next best $q^*_{k+1}(s, a)$; find θ_{k+1} such that $Q^\pi = q^*_{k+1}$

Pros

- It works

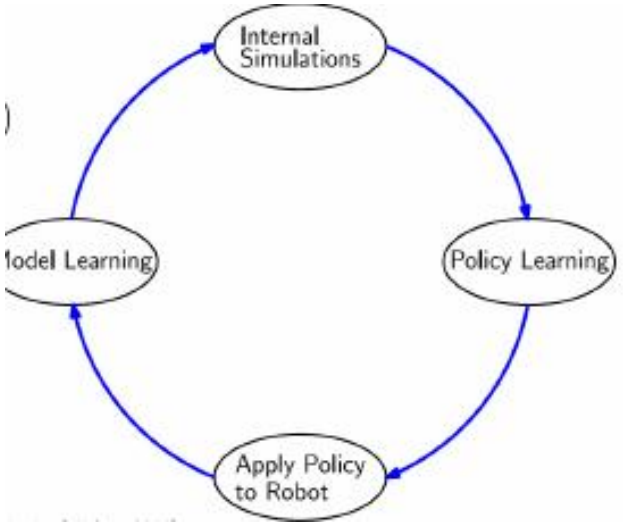
Cons

- Requires tons of examples
- Optimization process difficult to tune:
  - Learning rate difficult to adjust
  - Regularization (e.g. using KL divergence) badly needed and difficult to adjust

Direct Policy Search. The model-based approach

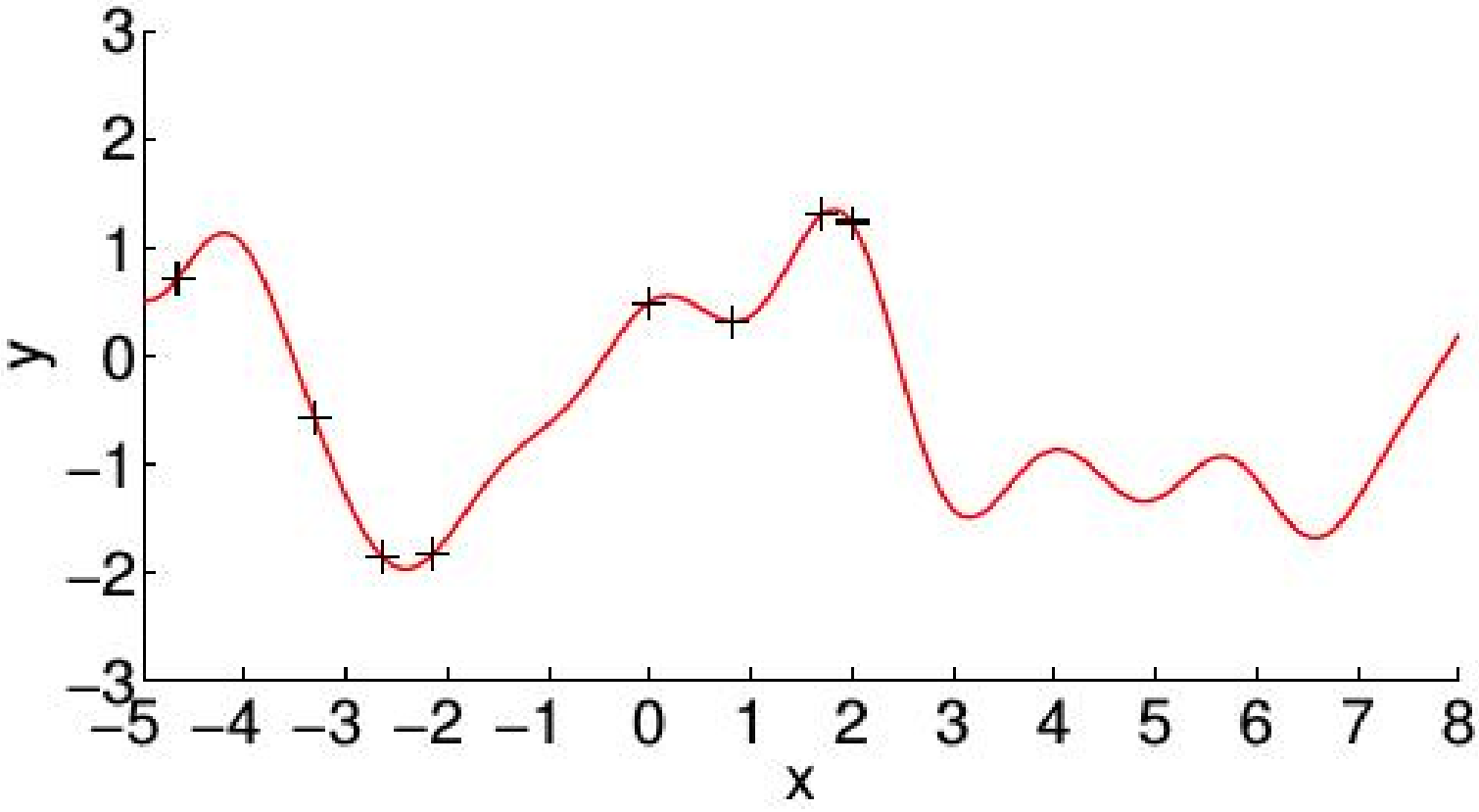
Algorithm

1. Use data $\tau_i = (s_{i,t}, a_{i,t})_{t=1}^{T}$ to learn a forward model $\hat{p}(s'|s,a)$
2. Use the model as a simulator
   (you need the estimation, **and the confidence of the estimation**, for exploration)
3. Optimize policy
4. (Use policy on robot and improve the model)



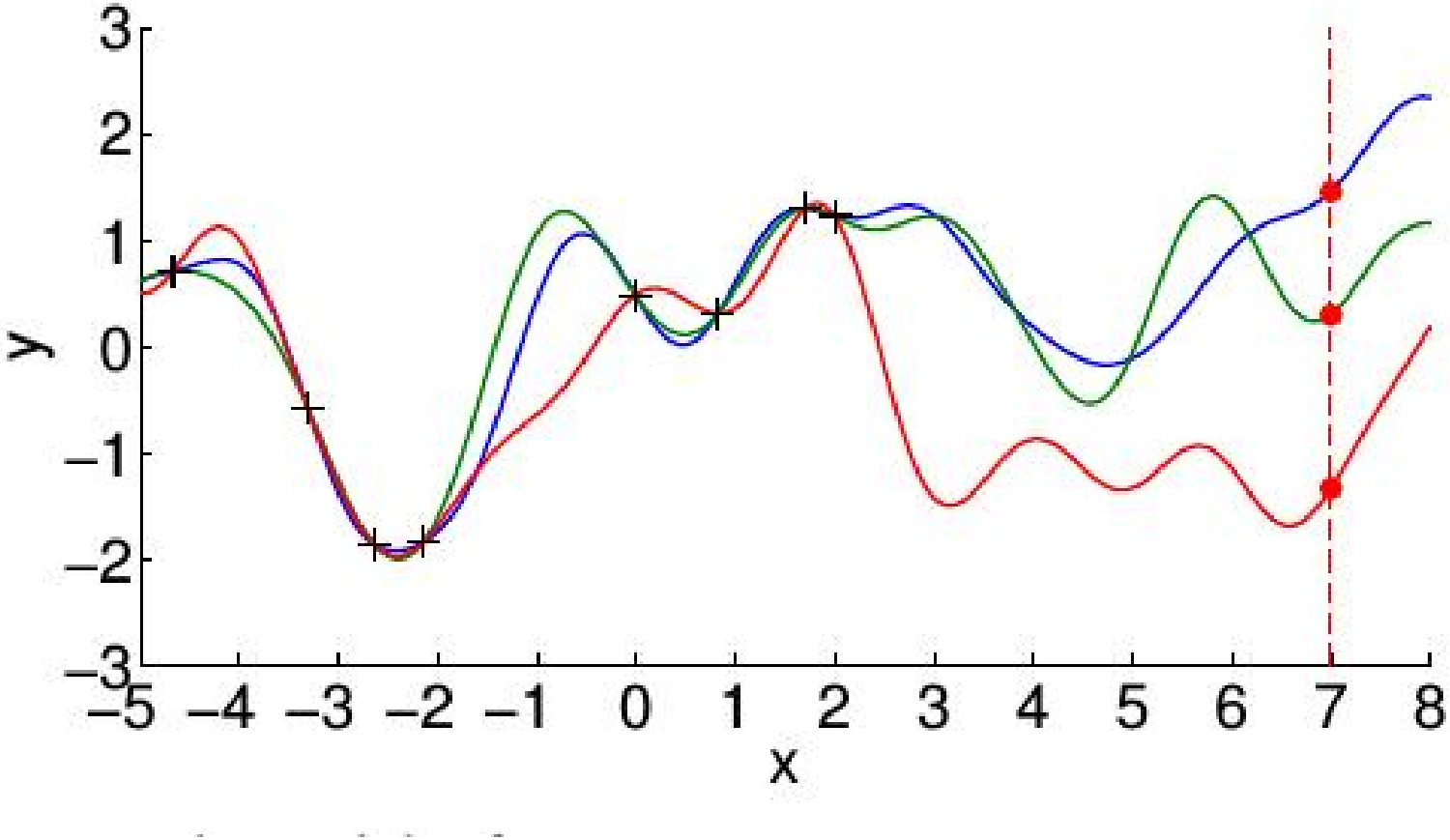
Learning the model

Modeling



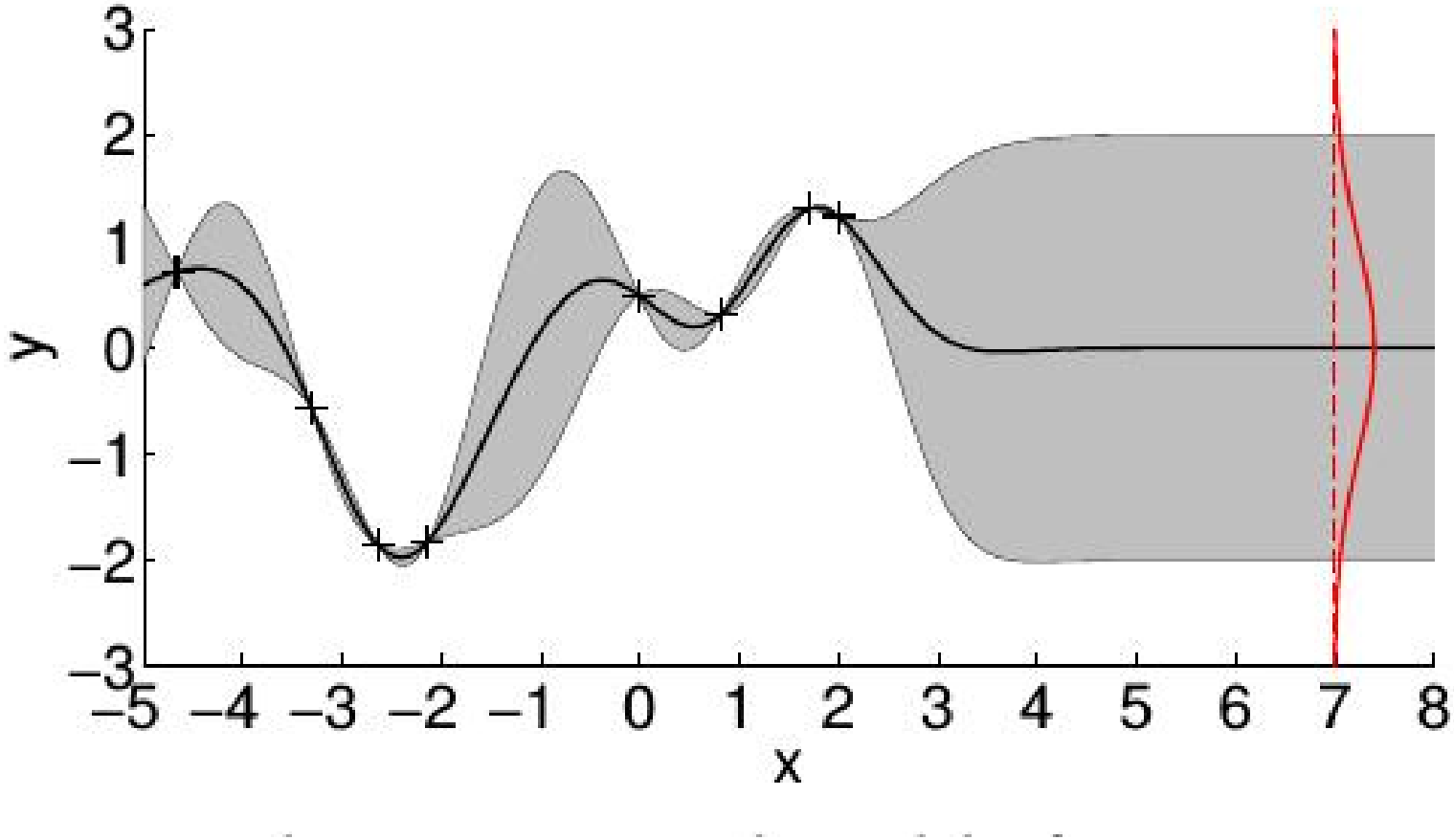
Learning the model

Modeling and predicting



Learning the model

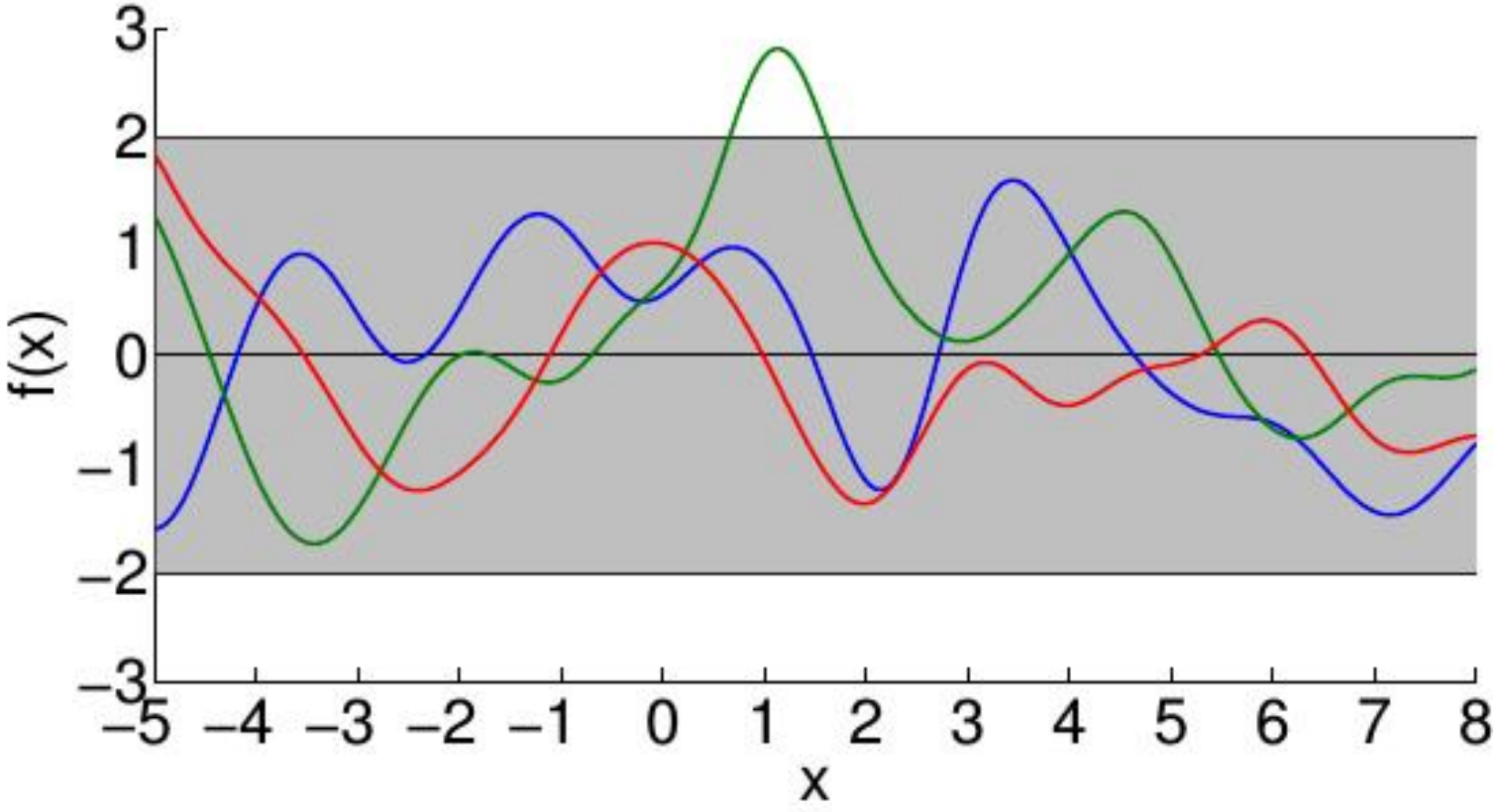
Modeling



When optimizing a model: very useful to have a measure of uncertainty on the prediction

Learning the model, 2

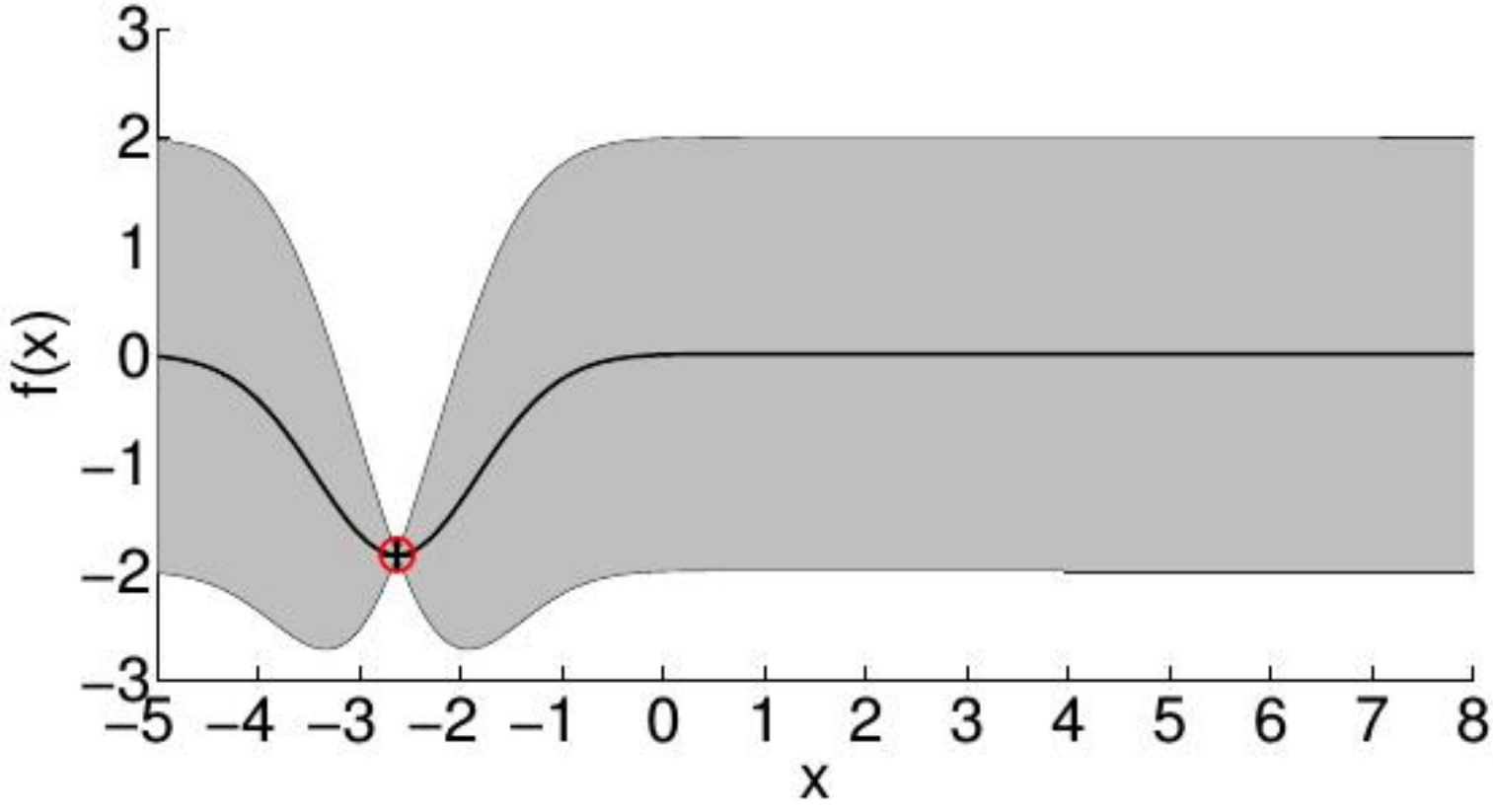
Gaussian Processes http://www.gaussianprocess.org/



Prior belief about the function

Learning the model, 2

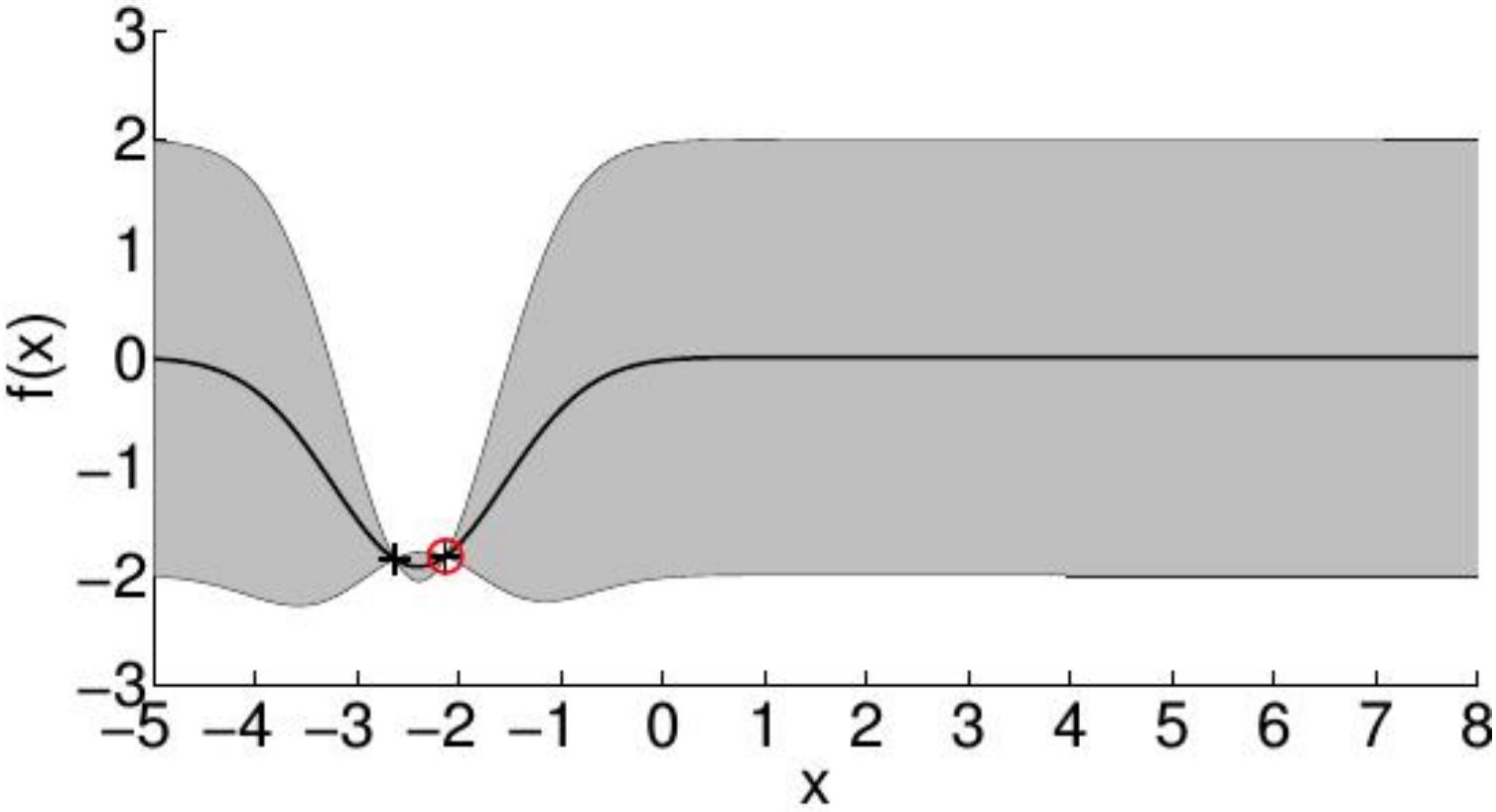
Gaussian Processes http://www.gaussianprocess.org/



Posterior belief about the function

Learning the model, 2

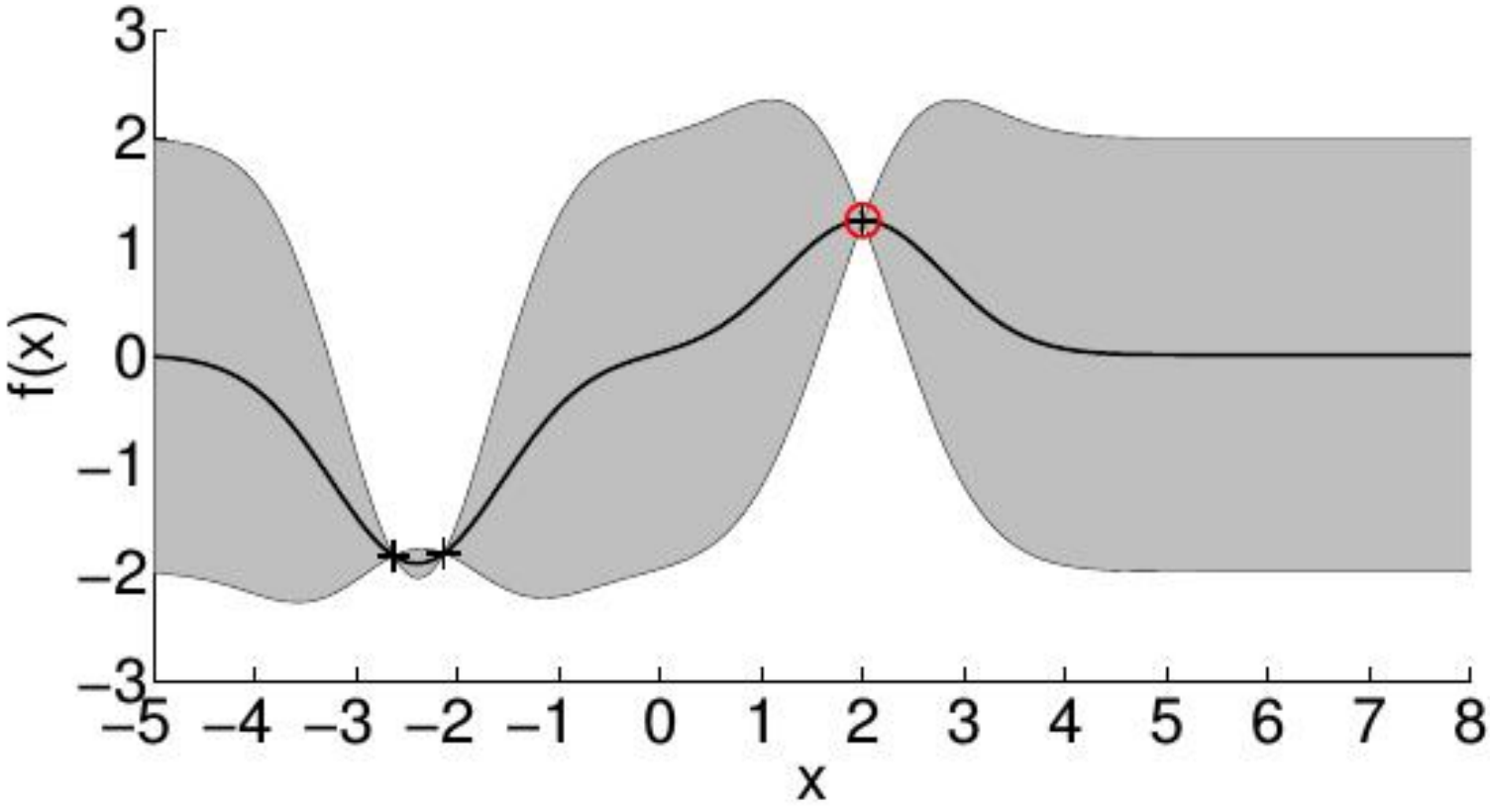
Gaussian Processes http://www.gaussianprocess.org/



Posterior belief about the function

Learning the model, 2

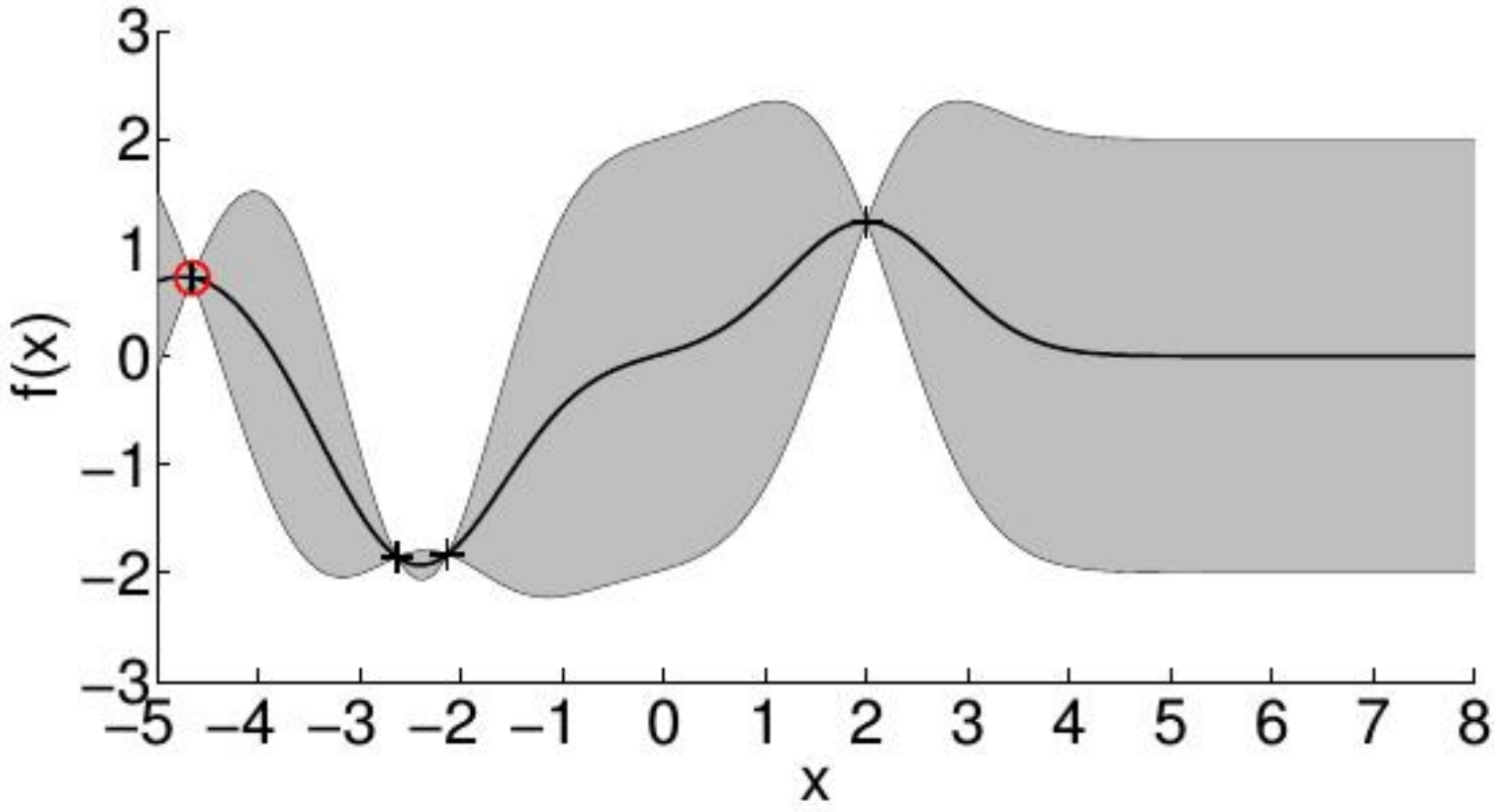
Gaussian Processes http://www.gaussianprocess.org/



Posterior belief about the function

Learning the model, 2

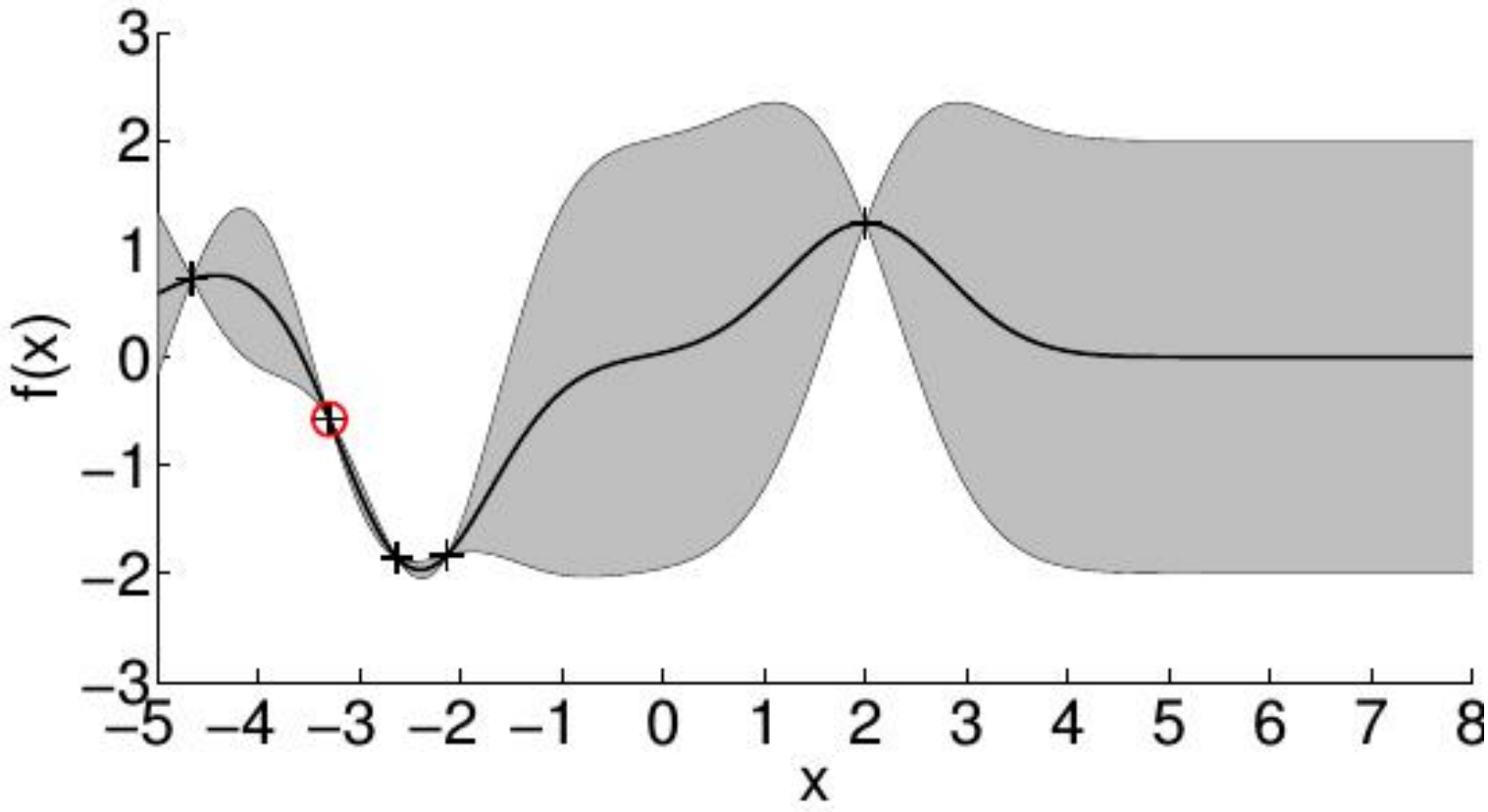
Gaussian Processes http://www.gaussianprocess.org/



Posterior belief about the function

Learning the model, 2

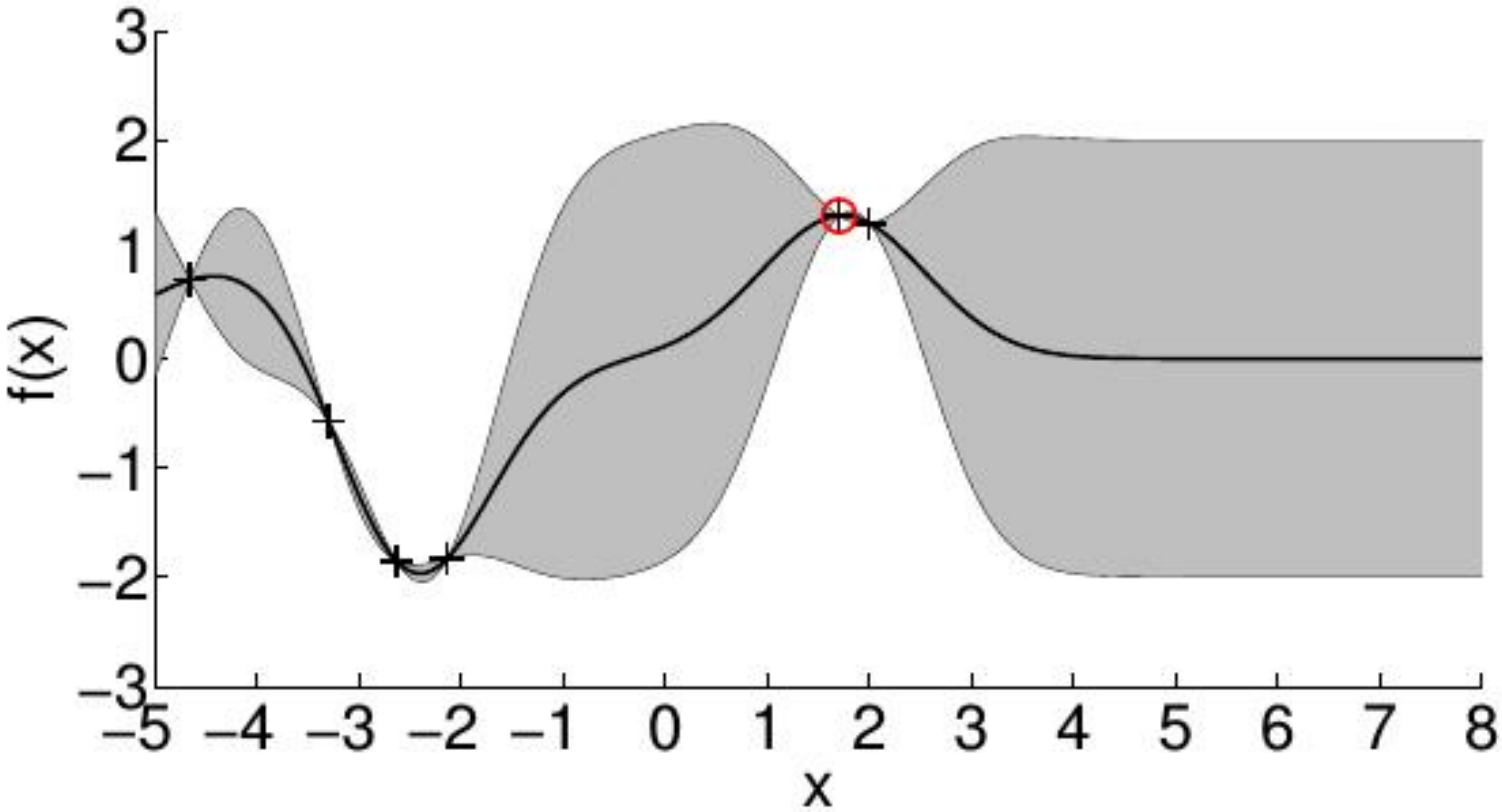
Gaussian Processes http://www.gaussianprocess.org/



Posterior belief about the function

Learning the model, 2

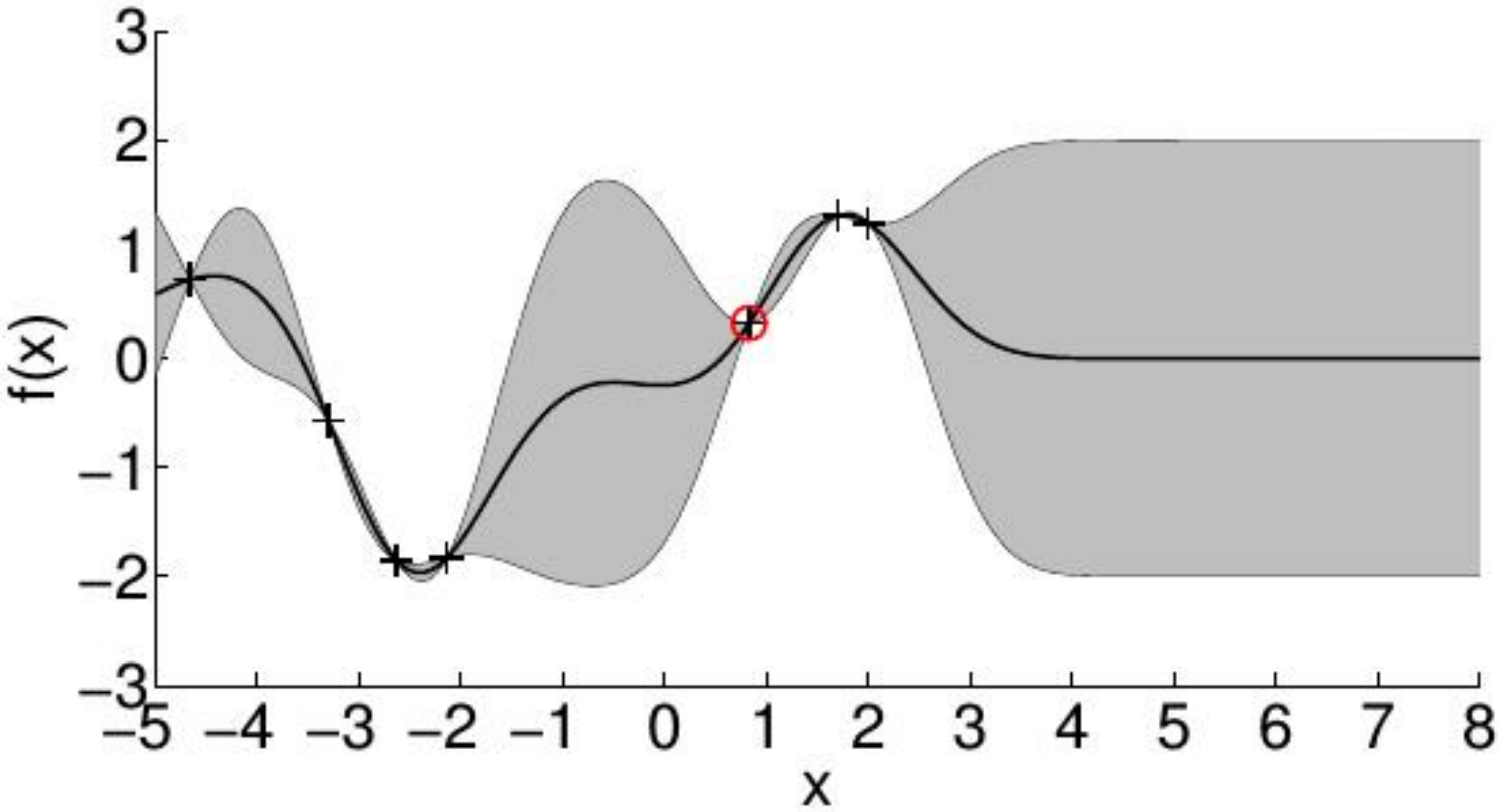
Gaussian Processes http://www.gaussianprocess.org/



Posterior belief about the function

Learning the model, 2

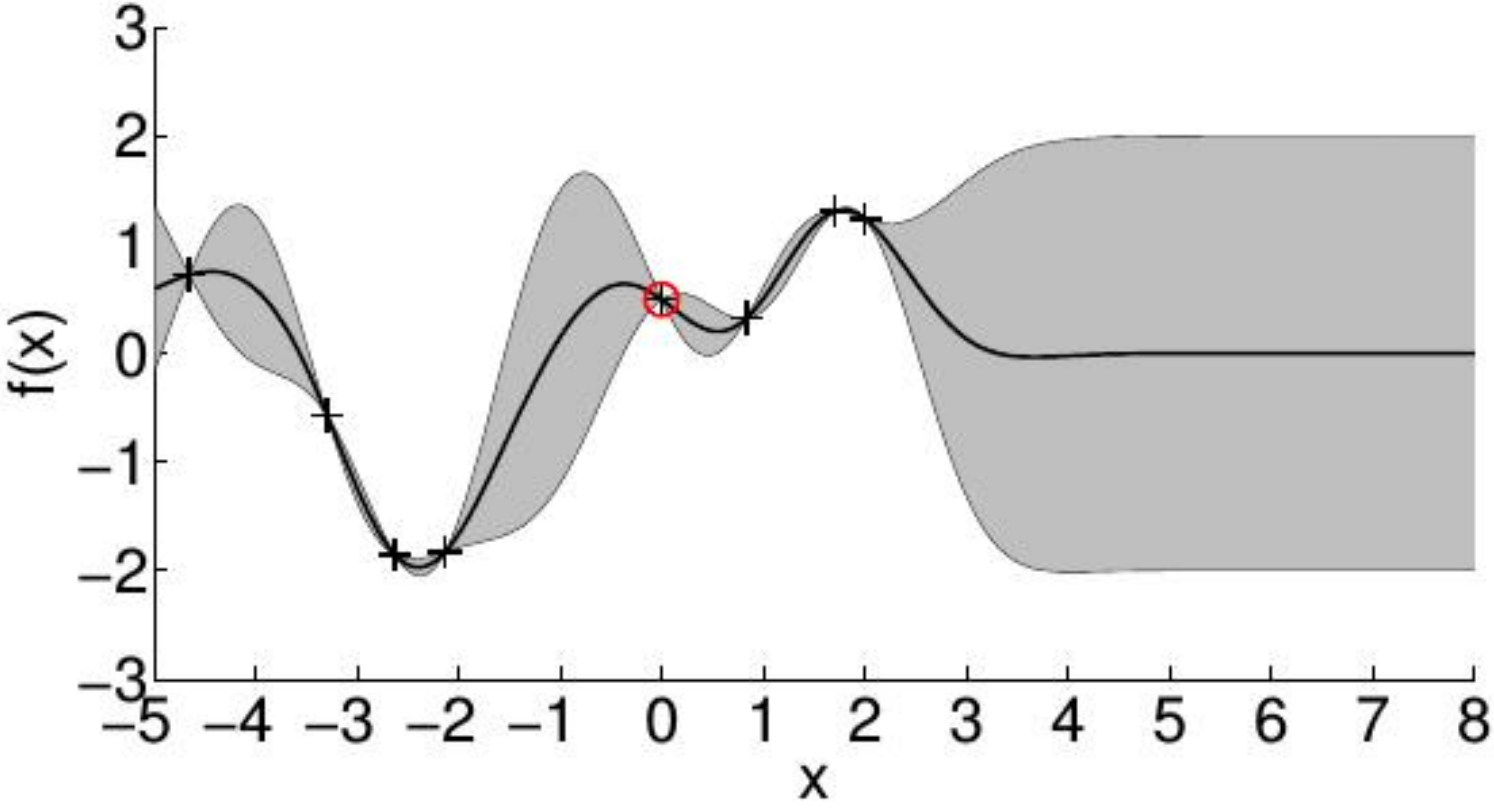
Gaussian Processes http://www.gaussianprocess.org/



Posterior belief about the function

Learning the model, 2

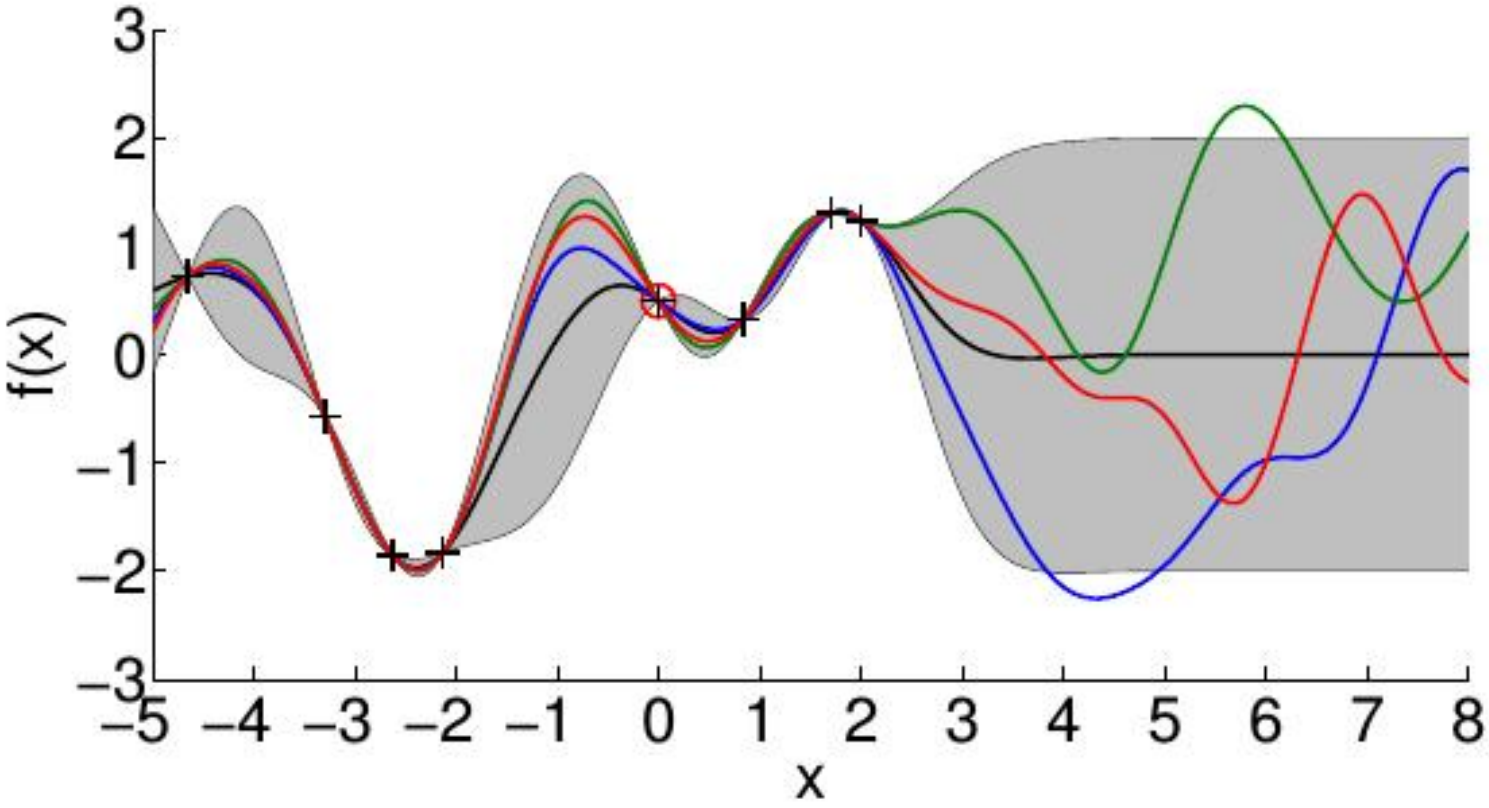
Gaussian Processes http://www.gaussianprocess.org/



Posterior belief about the function

Learning the model, 2

Gaussian Processes http://www.gaussianprocess.org/



Posterior belief about the function

Computing the gradient

Given

- Forward model

$$s_{t+1} = f(s_t, a_t)$$

- Differentiable policy

$$a = \pi(s_t, \theta)$$

It comes

$$V(\theta) = \sum_t \gamma^t r_{t+1}$$

Exact gradient computation

$$\frac{\partial V(\theta)}{\partial \theta} = \sum_t \gamma^t \frac{\partial r_{t+1}}{\partial \theta}$$

$$= \sum_t \gamma^t \frac{\partial r_{t+1}}{\partial s_{t+1}} \cdot \frac{\partial s_{t+1}}{\partial \theta}$$

$$= \sum_t \gamma^t \frac{\partial r_{t+1}}{\partial s_{t+1}} \left(\frac{\partial s_{t+1}}{\partial s_t} \cdot \frac{\partial s_t}{\partial \theta} + \frac{\partial s_{t+1}}{\partial a_t} \cdot \frac{\partial a_t}{\partial \theta} \right)$$

Model-based Direct Policy Search, summary

Algorithm

- Learn a model (prediction and confidence interval)
- Derive the gradient of the policy return
- Optimize it standard gradient optimization, e.g. BFGS

Pros

- Sample efficient (= does not require tons of examples)
- Fast (standard gradient-based optimization)
- Best ever results on some applications (pendulum on a car, picking up objects, controlling throttle valves)

Cons

- Gaussian processes (modelling also the confidence interval) hardly scale up: in $O(n^3)$, with n the number of examples
- Require specific parametrizations of the policy and the reward function
- Only works if the model is good (otherwise, disaster)

Evolutionary Robotics

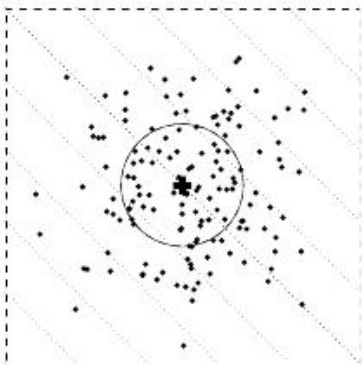
1. Select the search space Θ
2. Define the objective function $\mathcal{F}(\theta)$ in simulation or in-situ
   Sky is the limit: controller; morphology of the robot; co-operation of several robots...
3. Optimize: Evolutionary Computation (EC) and variants
4. Test the found solution reality gap

Covariance-Matrix-Adaptation-ES

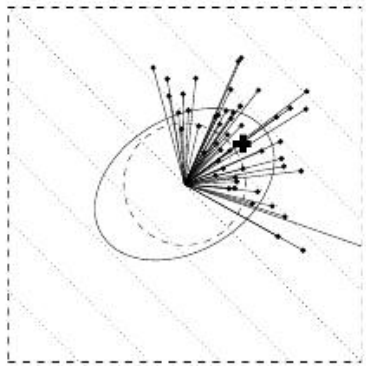
Hansen-Ostermeier, 2001; Auger-Hansen, 2010-2017

$$\theta \sim \mathcal{D}_k = \mathcal{N}(\mu_k, \Sigma_k)$$

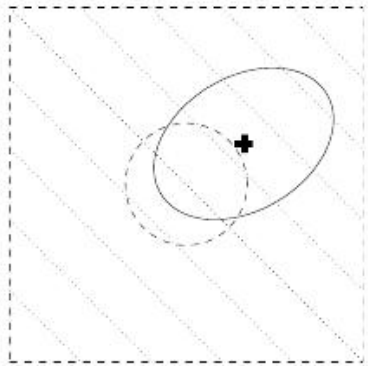
- easy to adapt μ_k
- Computationally heavy to adapt Σ_k
- does not scale up to high dimensions (> 200)



$\boldsymbol{y}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{C})\,, \quad \mathbf{C} = \mathbf{I}$
$\boldsymbol{x}_i = \boldsymbol{m} + \sigma\, \boldsymbol{y}_i, \quad \sigma = 1$



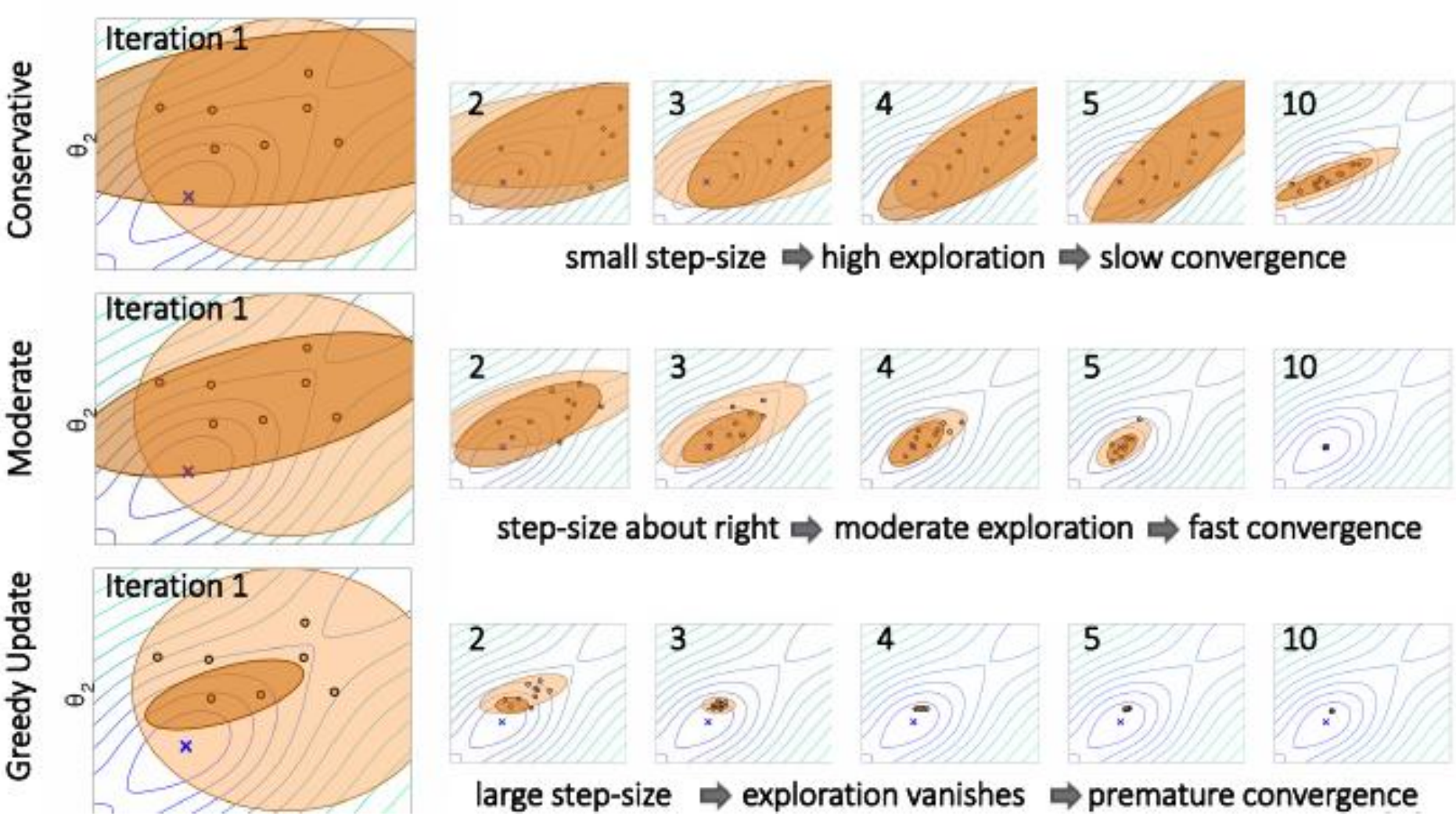
$\mathbf{C}_\mu = \frac{1}{\mu} \sum \boldsymbol{y}_{i:\lambda} \boldsymbol{y}_{i:\lambda}^{\mathrm{T}}$
$\mathbf{C} \leftarrow (1-1) \times \mathbf{C} + 1 \times \mathbf{C}_\mu$



$\boldsymbol{m}_{\text{new}} \leftarrow \boldsymbol{m} + \frac{1}{\mu} \sum \boldsymbol{y}_{i:\lambda}$

- Invariances under monotonous transform of optimization criterion and affine transf. of Θ.
- A particular case of Information Geometry Optimization

Effects of step size



Search Space, 1

Neural Nets

- Universal approximators; continuity; generalization hoped for.
- Fast computation
- Can include priors in the structure
- Feedforward architecture: reactive policy
- Recurrent architecture: internal state

  encoding memory (fast vanishing)

Critical issues

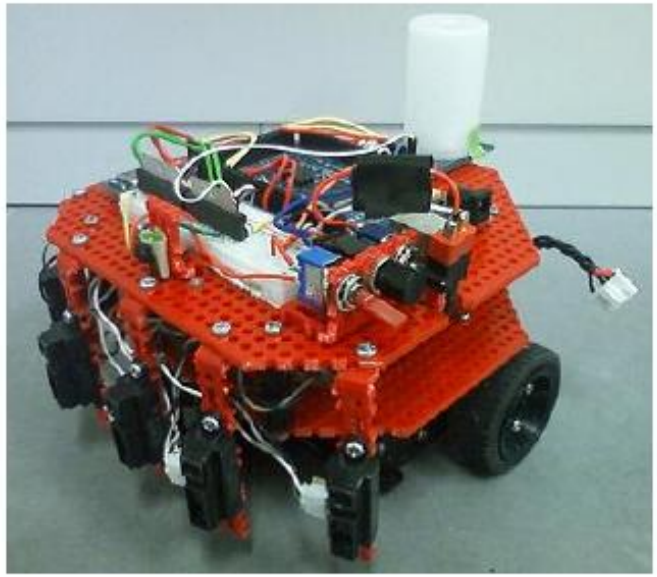
- Non-parametric optimization much more difficult

Other options

- Finite state automaton (find states; write rules; optimize thresholds...)
  The Braitenberg controller.
- Genetic programming (optimization of programs)

Example: Swarm robots moving in column formation

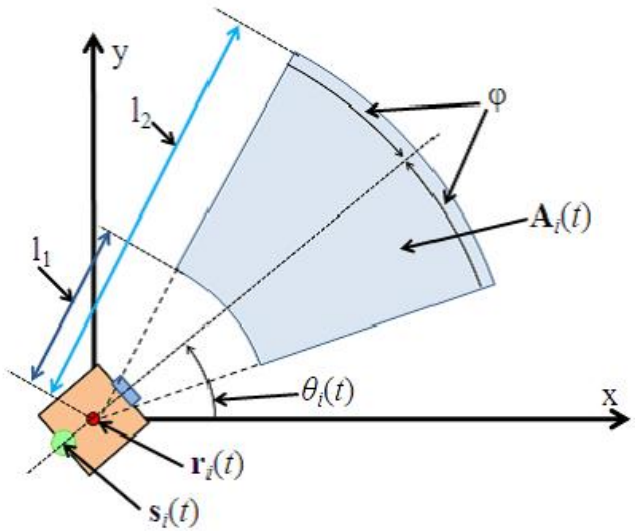
Robot



Robotic swarm, 2

Representation

Constants		
	l1	blind zone
	l2	sensor range
	ϕ	Vision angular range
Variables(t)		
	$r(t), s(t)$	positions
	$\theta(t)$	angular direction



Example of a (almost manual) controller

CONTROLLER OF A ROBOT

Info. from the image sensors	Info. from the IR sensors		
	$0 \leqq x_{\text{IR}} < \beta_0$	$\beta_0 \leqq x_{\text{IR}} < \beta$	$\beta \leqq x_{\text{IR}}$
$0 \leqq x_{\text{image}} \leqq \alpha$	move backward or turn right	turn left	
$\alpha < x_{\text{image}} < (19 - \alpha)$	move backward or turn right	stop	move forward
$\alpha \leqq x_{\text{image}} \leqq 19$	move backward or turn right	turn right	
preceding robot NOT FOUND	move backward or turn right	move forward	

Toward defining $\mathcal{F}$

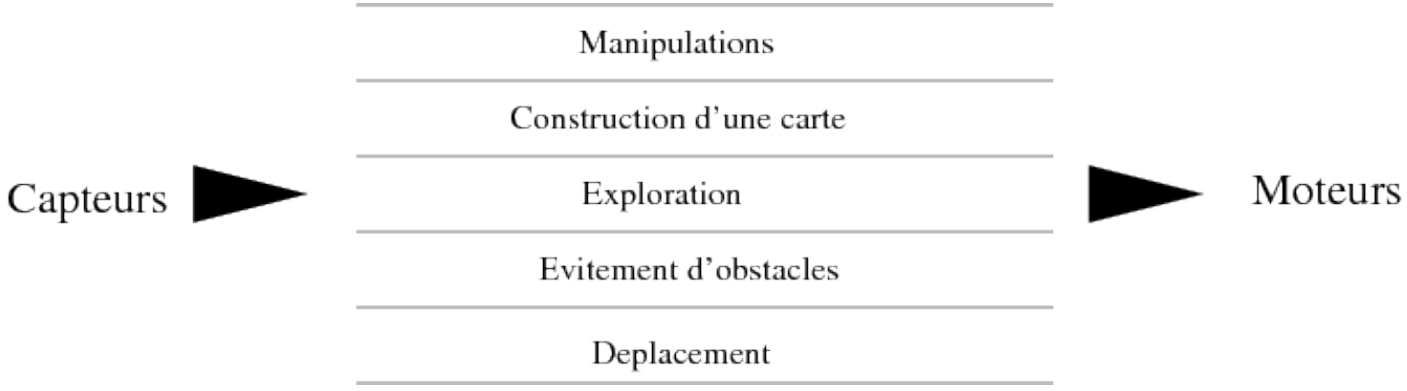
- The i-th robot follows the k-th robot at time t iff the center of gravity of k belongs to the perception range of i ($\mathbf{s}_k(t) \in \mathbf{A}_i(t)$).
- The i-th robot is a leader if i) it does not follow any other robot; ii) there exists at least one robot following it.
- A column is a subset $\{i_1, \ldots i_K\}$ such that robot i_{k+1} follows robot i_k and robot i_1 is a leader.
- A deadlock is a subset $\{i_1, \ldots i_K\}$ such that robot i_{k+1} follows robot i_k and robot i_1 follows robot i_K.

Optimization criterion

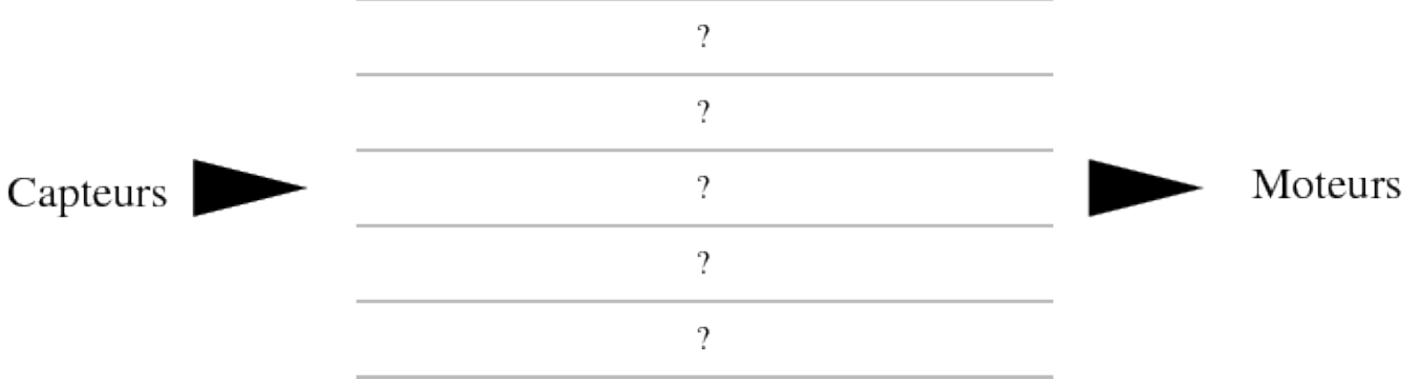
Brooks 89-01

The promise: no need to decompose the goal

- Behavioral robotics — hand crafted decomposition

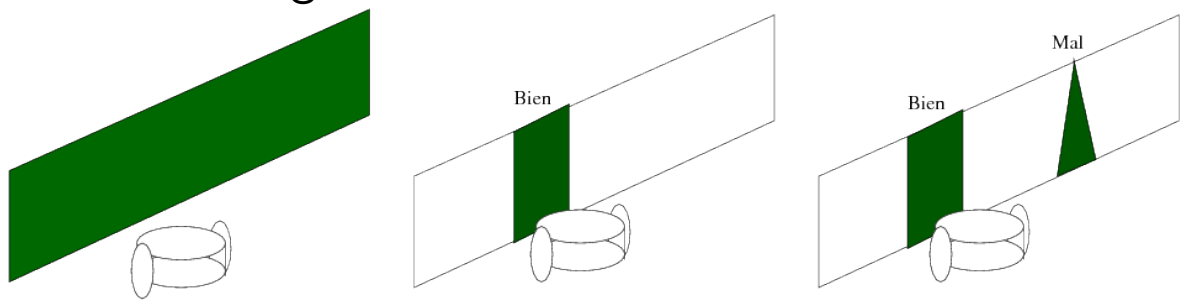


- Evolutionary robotics — emergence of a structure



In practice: fitness shaping

- All initial (random) individuals are just incompetent
- Fitness landscape: Needle in the Haystack ? (doesn't work)
- Start with something simple
- Switch to more complex *during evolution*
- Example: visual recognition



Optimization criterion, 2

- Fonctional vs behavioral

  state of controller vs distance walked
- Implicit vs explicit

  Survival vs Distance to socket
- Internal vs external information

  Sensors, ground truth
- Co-evolution: e.g. predator/prey

  performance depends on the other robots

State of art

- Standard: function, explicit, external variables
- In-situ: behavioral, implicit, internal variables
- Interactive: behavioral, explicit, external variables

Optimization criterion, 3

Fitness shaping

- Obstacle avoidance
- Obstacle avoidance, and move !
- Obstacle avoidance, and (non circular) move !!

Finally

Floreano Nolfi 2000

$$\mathcal{F}(\theta) = \int_{T_{exp.}} A(1 - \sqrt{\Delta B})(1 - i)$$

- A sum of wheel speed $r_i \in [-0.5, 0.5]$

  $\rightarrow$ move
- $\Delta B = |r_1 + r_2|$

  $\rightarrow$ ahead
- i maximum (normalised) of sensor values

  $\rightarrow$ obstacle avoidance

Behavioral, internal variables, explicit

Result analysis

- First generations
  - Most rotate
  - Best ones slowly go forward
  - No obstacle avoidance
  - Perf. depends on starting point
- After $\approx$ 20 gen.
  - Obstacle avoidance
  - No rotation
- Thereafter, gradually speed up

Result analysis, 2

- Max. speed 48mm/s (true max = 80)

  Inertia, bad sensors

- Never stuck in a corner

  contrary to Braitenberg

Going further

- Changing environment
- Changing robotic platform

Limitations

- From simulation to real-world — Reality gap !
- Opportunism of evolution
- Roboticists not impressed...

Carl Sims

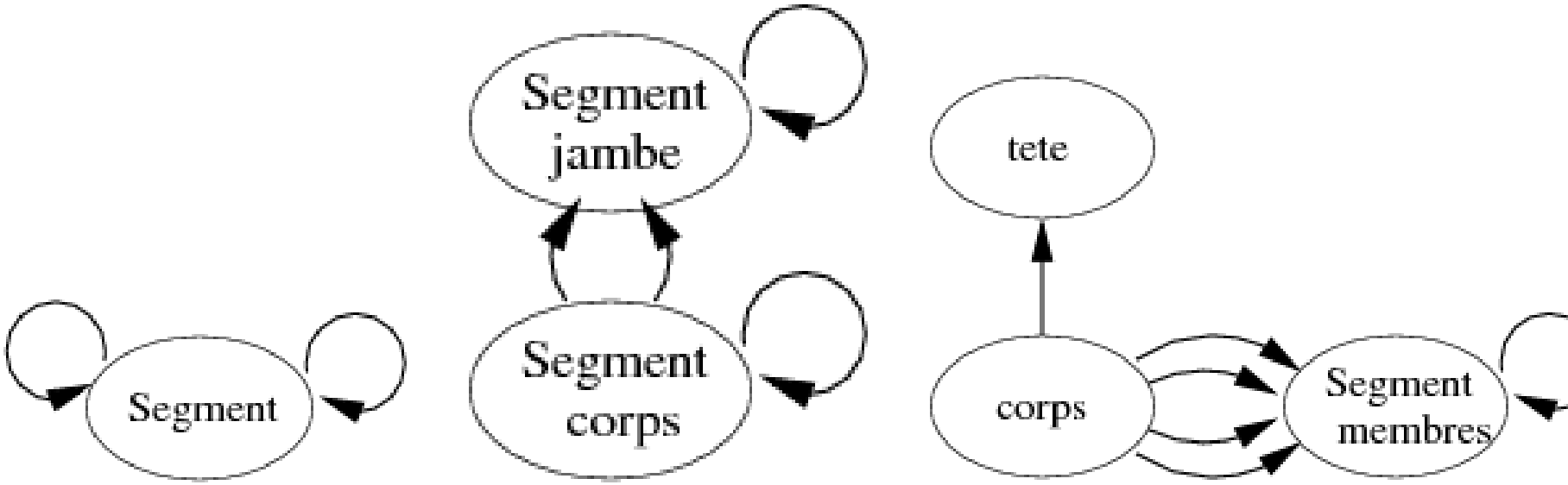
Goal

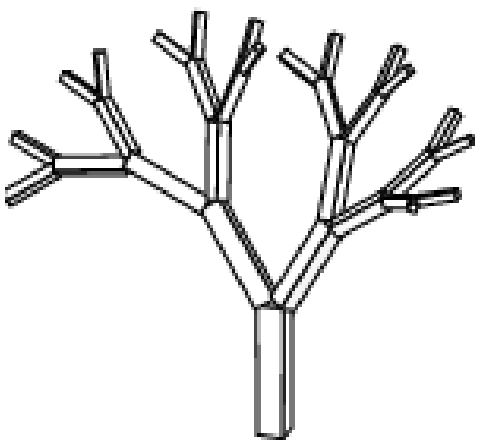
- Evolve both morphology and controller
- using a grammar (oriented graph)
- Heavy computational cost

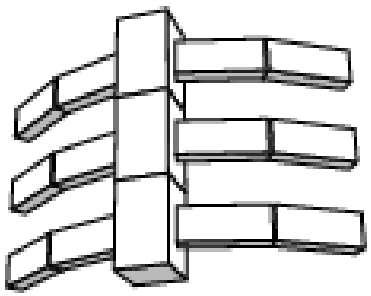
  simulation, several days on Connection Machine – 65000 proc.
- Evolving locomotion (walk, swim, jump)
- and competitive co-evolution (catch an object)

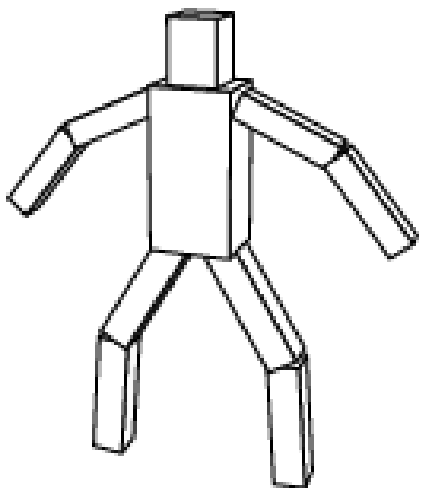
The creatures

Karl Sims, 1994









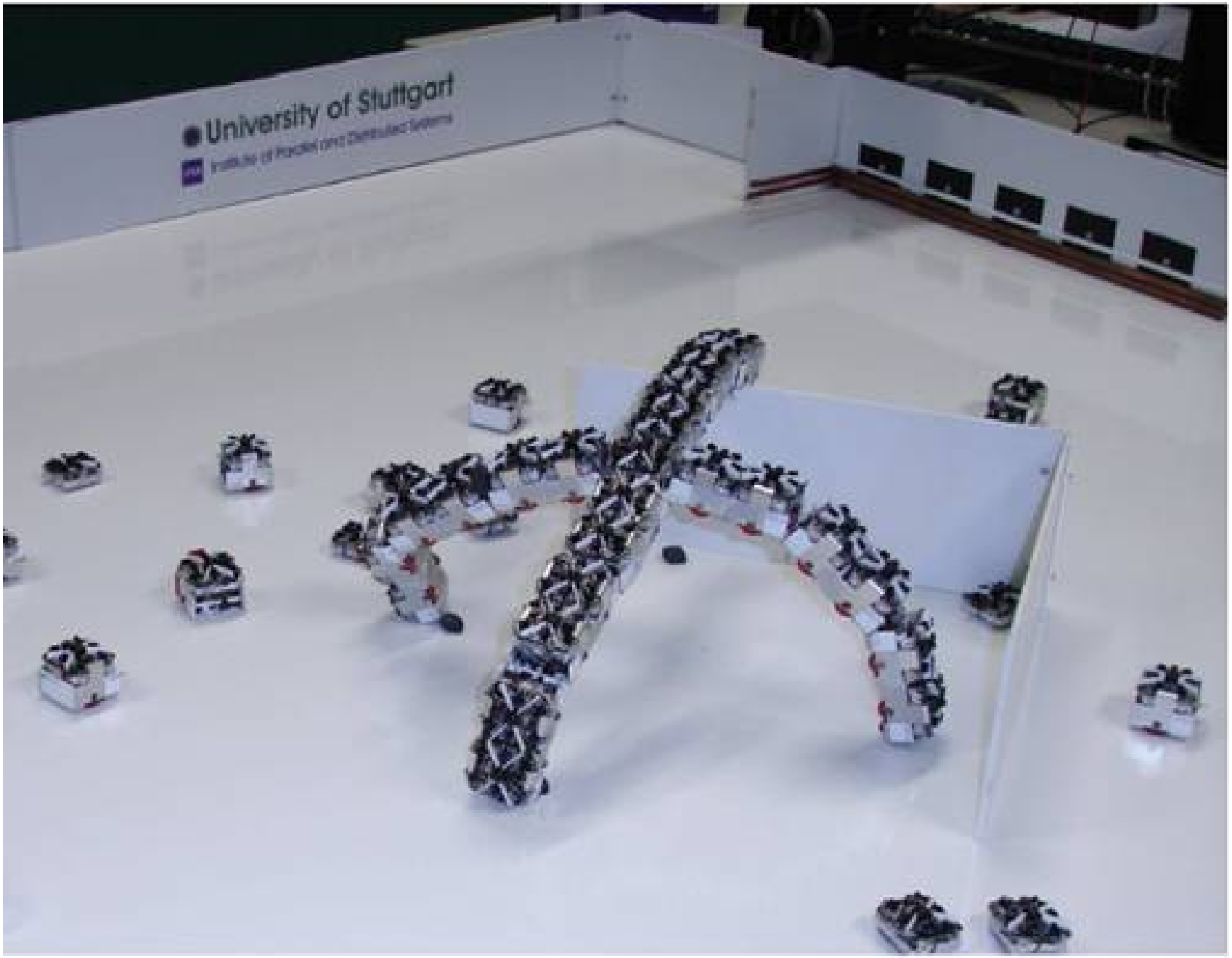
Video: https://www.youtube.com/watch?v=JBgG_VSP7f8

Reset-Free Trial and Error

Jean-Baptiste Mouret, 17

https://www.youtube.com/watch?v=IqtyHFrb3BU

Intrinsic rewards, swarm robotics



https://www.youtube.com/watch?v=btNLWKdngq4

Internal rewards

Delarboulas et al., PPSN 2010

Requirements

1. No simulation
2. On-board training
   - Frugal (computation, memory)
   - No ground truth
3. Providing "interesting results"

"Human – robot communication"

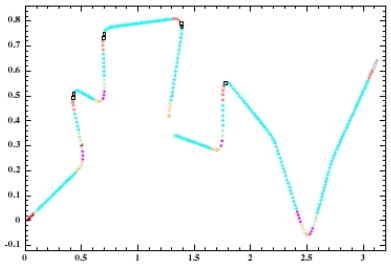
Goal: self-driven Robots : **Defining instincts**

Starting from (almost) nothing

Robot $\equiv$ a data stream

$$t \rightarrow x[t] = (sensor[t], motor[t])$$

$$\text{Trajectory} = \{x[t], t = 1 \ldots T\}$$



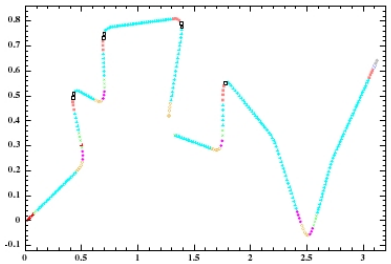
Robot trajectory

Starting from (almost) nothing

Robot $\equiv$ a data stream

$$t \rightarrow x[t] = (sensor[t], motor[t])$$

Trajectory $= \{x[t], t = 1 \ldots T\}$



Robot trajectory

Computing the quantity of information of the stream
Given $x_1, \ldots x_n$, visited with frequency $p_1 \ldots p_n$,

$$Entropy(trajectory) = -\sum_{i=1}^{n} p_i \log p_i$$

Conjecture

Controller quality $\propto$ Quantity of information of the stream

Building sensori-motor states

Avoiding trivial solutions...

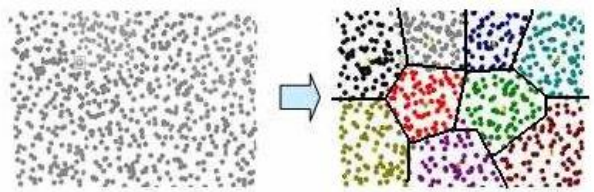
If sensors and motors are continuous / high dimensional

- then all vectors $x[t]$ are different
- then $\forall i, p_i = 1/T$; $Entropy = \log T$

... requires generalization

From the sensori-motor stream

to clusters



sequence of points in $\mathbb{R}^d$

sensori-motor states

Trajectory $\rightarrow$

$x_1 x_2 x_3 x_1 \ldots$

Clusters in sensori-motor space ($\mathbb{R}^2$)

Clustering

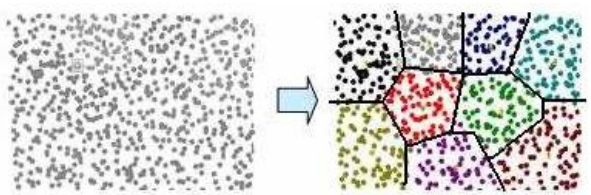
k-Means

1. Draw k points $x[t_i]$
2. Define a partition $\mathcal{C}$ in k subsets C_i *Voronoï cells*

$$C_i = \{x / d(x, x[t_i]) < d(x, x[t_j]), j \neq i\}$$

ϵ-Means

1. Init : $\mathcal{C} = \{\}$ *Initial site list*
2. For $t = 1$ to T *loop on trajectory*
   - If $d(x[t], \mathcal{C}) > \epsilon$, $\mathcal{C} \leftarrow \mathcal{C} \cup \{x[t]\}$



Curiosity Instinct

Search space

- Neural Net, 1 hidden layer.

Definition

- Controller F + environment $\rightarrow$ Trajectory
- Apply Clustering on Trajectory
- For each C_i, compute its frequency p_i

$$\mathcal{F}(F) = -\sum_{i=1}^{n} p_i * \log(p_i)$$

Curiosity instinct: Maximizing Controller IQ

Properties

- Penalizes inaction: a single state $\rightarrow$ entropy $= 0$
- Robust w.r.t. sensor noise (outliers count for very little)
- Computable online, on-board (use ϵ-clustering)
- Evolvable onboard

Limitations: does not work if

- Environment too poor
  (in desert, a single state $\rightarrow$ entropy $= 0$)
- Environment too rich
  (if all states are distinct, $\mathit{Fitness}(\text{controller}) = \log T$)

both under and over-stimulation are counter-effective.

From curiosity to discovery

Intuition

- An individual learns sensori-motor states ($x[t_i]$ center of C_i)
- The SMSs can be transmitted to offspring
- giving the offspring an access to "history"
- The offspring can try to "make something different"

$$\textbf{fitness(offspring)} = \textbf{Entropy(Trajectory(ancestors} \bigcup \textbf{offspring))}$$

NB: does not require to keep the trajectory of all ancestors.
One only needs to store $\{C_i, n_i\}$

From curiosity to discovery

Cultural evolution transmits genome + "culture"

1. parent = (controller genome, $(C_1, n_1), \ldots (C_K, n_K)$)
2. Perturb parent controller $\rightarrow$ offspring controller
3. Run the offspring controller and record $x[1], \ldots x[T]$
4. Run ϵ-clustering variant.

$$Fitness(offspring) = -\sum_{i=1}^{\ell} p_i \log p_i$$

ϵ-clustering variant

Algorithm

1. Init : $\mathcal{C} = \{(C_1, n_1), \ldots (C_K, n_K))\}$ *Initial site list*
2. For $t = 1$ to T *loop on trajectory*
   - If $d(x[t], \mathcal{C}) > \epsilon$, $\mathcal{C} \leftarrow \mathcal{C} \cup \{x[t]\}$
3. Define $p_i = n_i / \sum_j n_j$

$$Fitness(offspring) = -\sum_{i=1}^{\ell} p_i \log p_i$$

Limitation

In stochastic environments

- High entropy in highly stochastic regions

Intrinsic motivations, neuro-curiosity Oudeyer et al. 2005-2017

- More exploration $\rightarrow$ more data
- Are these data useful ?
- Yes if Reduction of error of learned forward model.

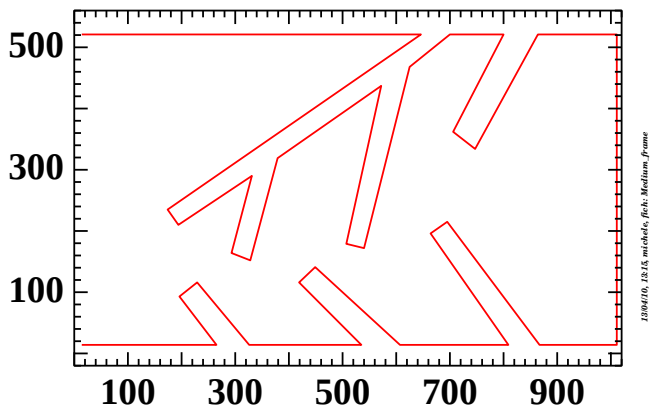
https://www.youtube.com/watch?v=bkv83GKYpkI

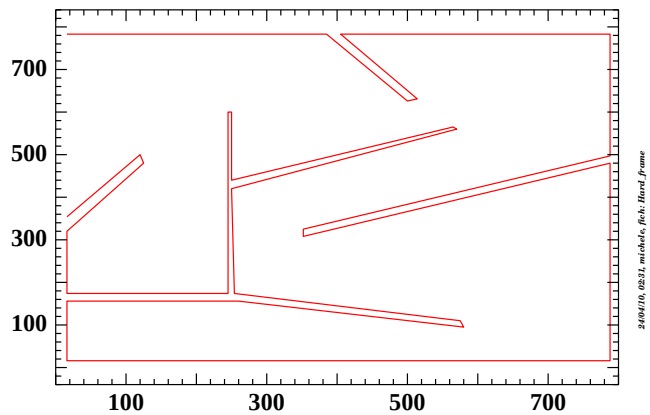
Validation

Experimental setting

Robot = Cortex M3, 8 infra-red sensors, 2 motors.
Controller space = ML Perceptron, 10 hidden neurons.

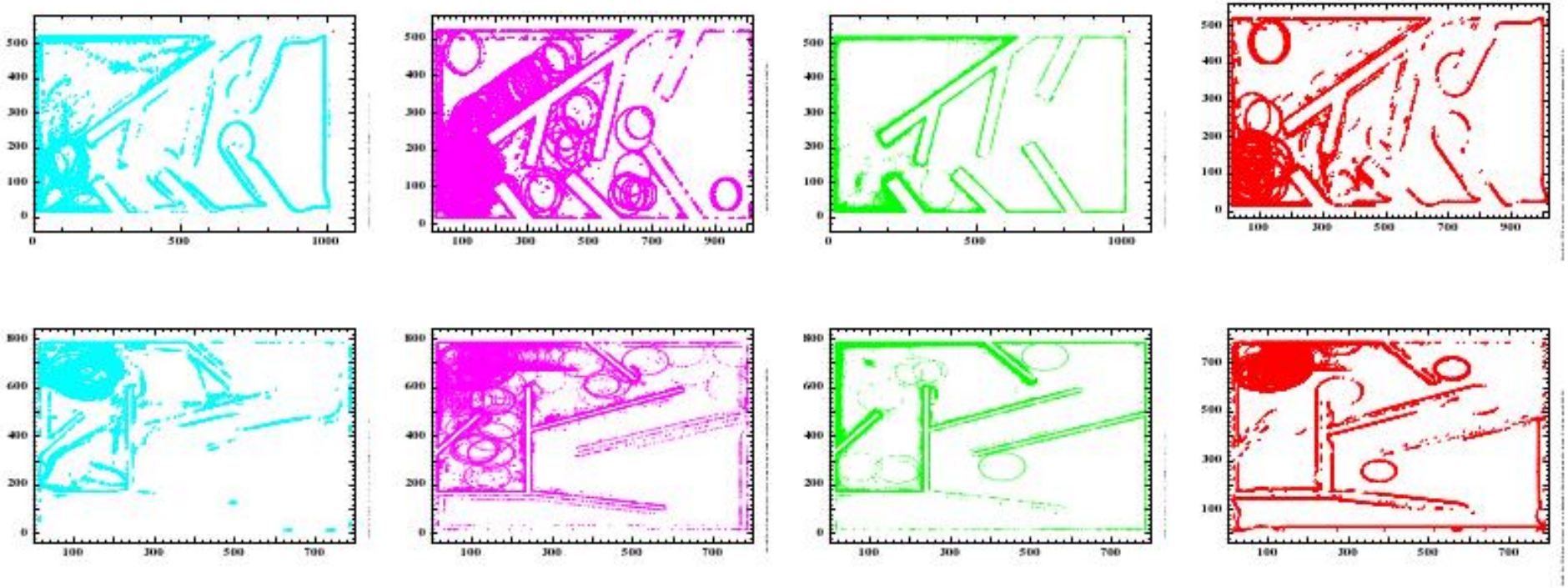
Medium and Hard Arenas





Validation, 2

Plot points in hard arena visited 10 times or more by the 100 best individuals.



Nolfi & Floreano | Lehman & Stanley | Curiosity | Discovery

PPSN 2010

Partial conclusions

Entropy-minimization

- computable on-board;
- yields "interesting" behavior
- needs stimulating environment

no need of prior knowledge/ground truth

Not covered

- Inverse Reinforcement Learning
  https://www.youtube.com/watch?v=VCdxqn0fcnE
- Programming by Feedback
- Deep Reinforcement Learning