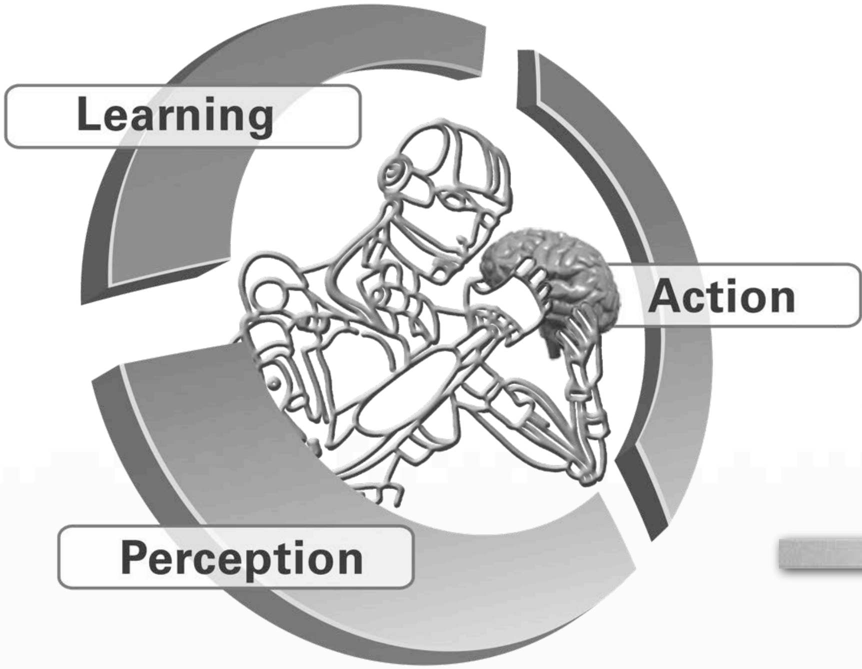


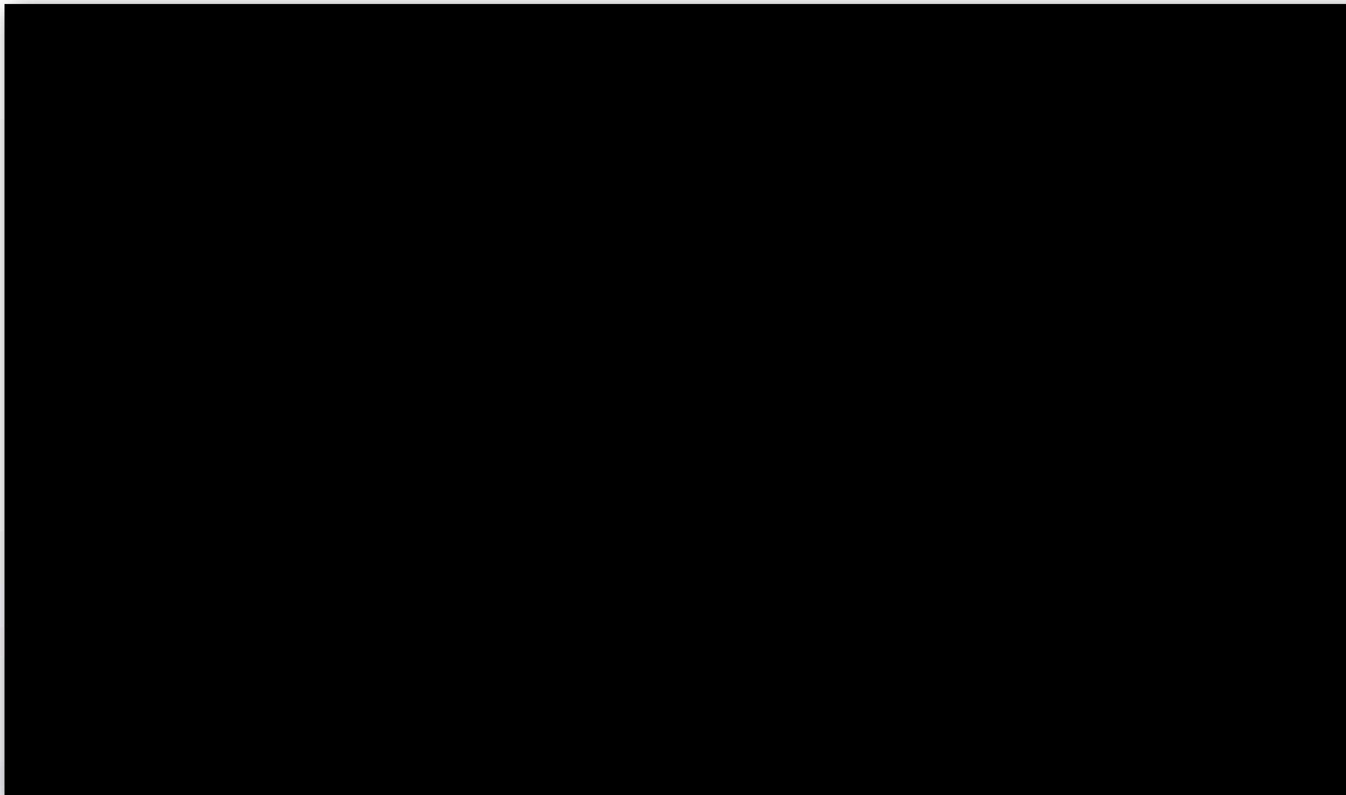
Robotics

Part II: From Learning Model-based Control to Model-free Reinforcement Learning

Stefan Schaal
Max-Planck-Institute for Intelligent Systems
Tübingen, Germany
&
Computer Science, Neuroscience, & Biomedical Engineering
University of Southern California, Los Angeles

sschaal@is.mpg.de
http://www-amd.is.tuebingen.mpg.de

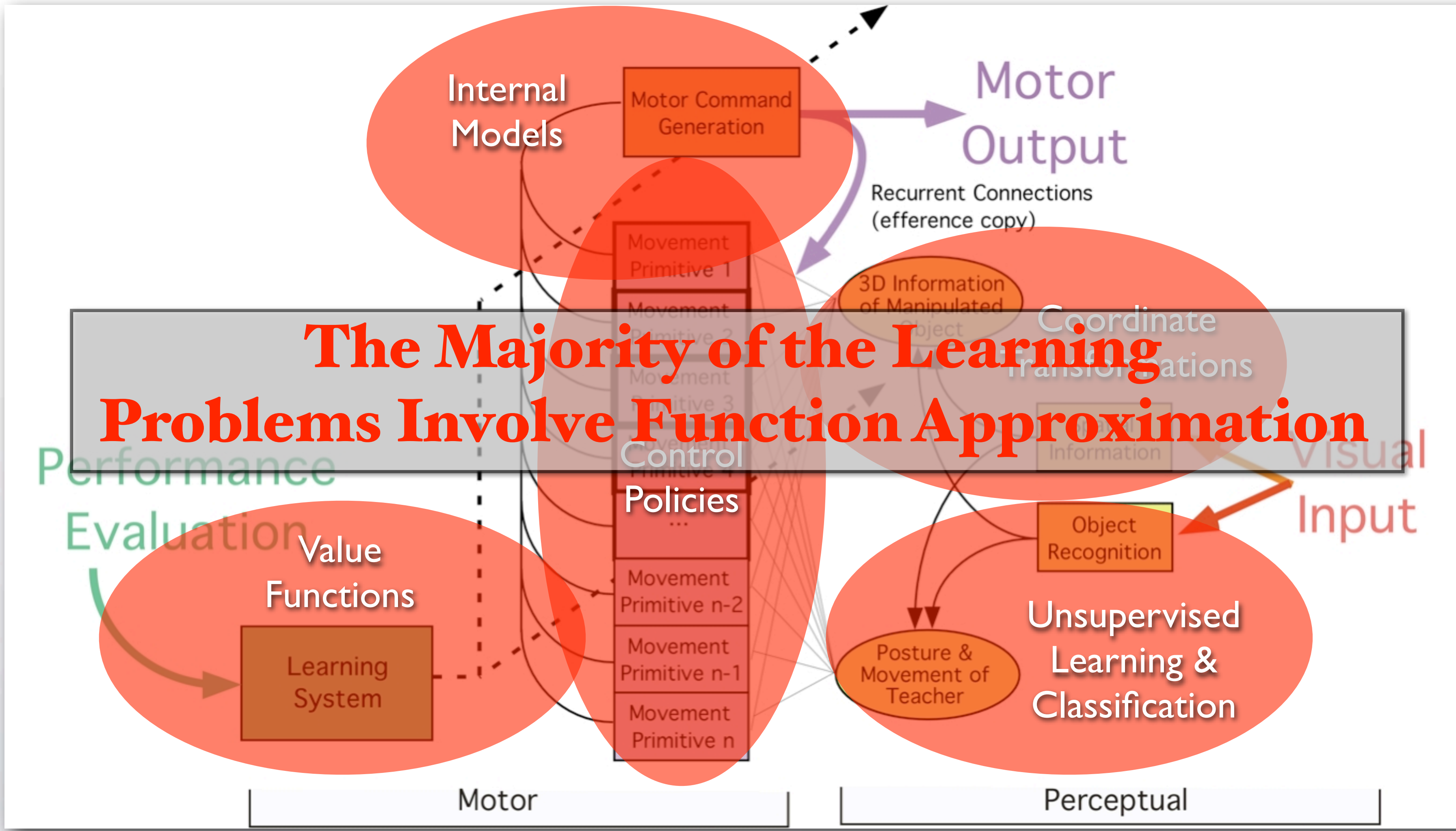
Where Did We Stop ...



Outline

- A Bit of Robotics History
- Foundations of Control
- Adaptive Control
- Learning Control
  - Model-based Robot Learning
  - Reinforcement Learning

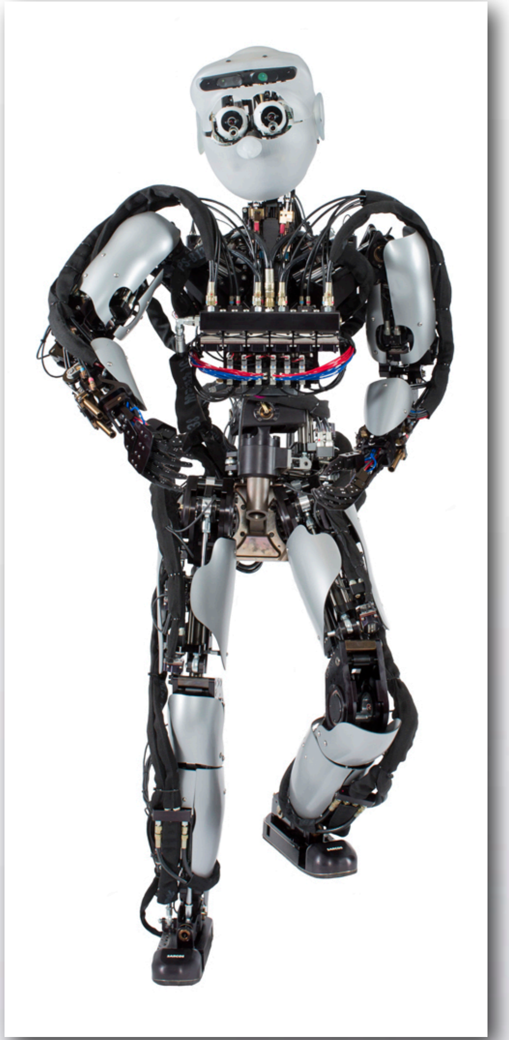
What Needs to Be Learned in Learning Control?



The Majority of the Learning Problems Involve Function Approximation

Characteristics of Function Approximation in Robotics

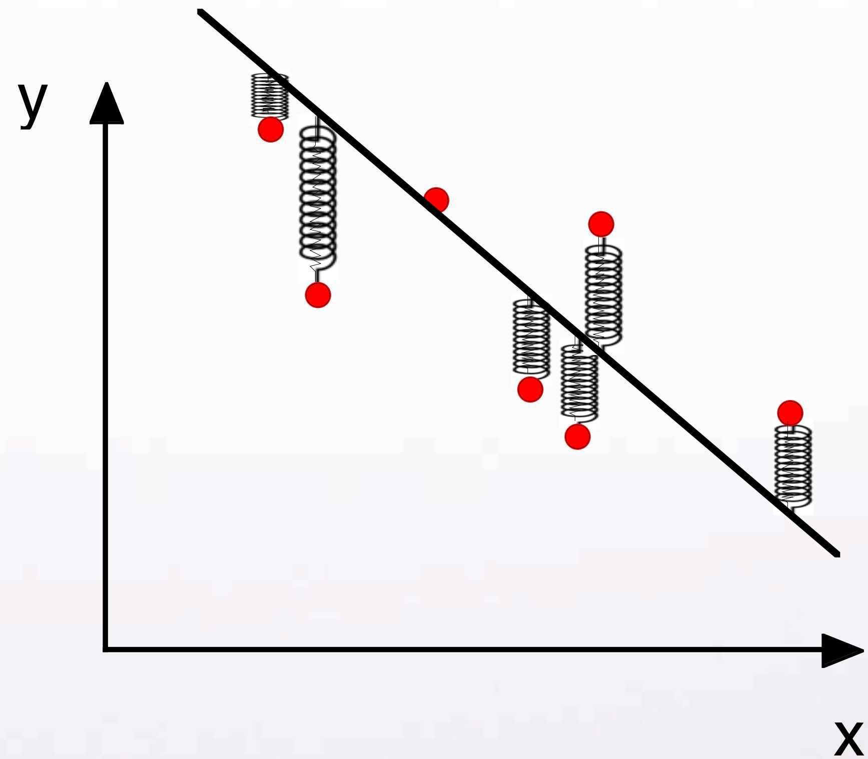
- Incremental Learning
  - large amounts of data
  - continual learning
  - to be approximated functions of growing and unknown complexity
- Fast Learning
  - data efficient
  - computationally efficient
  - real-time
- Robust Learning
  - minimal interference
  - hundreds of inputs



Linear Regression: One of the Simplest Function Approximation Methods

Recall the simple adaptive control model with: $f(x) = \theta x$

- find the line through all data points
- imagine a spring attached between the line and each data point
- all springs have the same spring constant
- points far away generate more "force" (danger of outliers)
- springs are vertical
- solution is the minimum energy solution achieved by the springs



Linear Regression: One of the Simplest Function Approximation Methods

- The data generating model:

$$y = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}} + w_0 + \varepsilon = \mathbf{w}^T \mathbf{x} + \varepsilon$$

$$\text{where } \mathbf{x} = \left[\mathbf{x}^T, 1\right]^T, \mathbf{w} = \begin{bmatrix} \tilde{\mathbf{w}} \\ w_0 \end{bmatrix}, E\{\varepsilon\} = 0$$

- The Least Squares cost function

$$J = \frac{1}{2}(\mathbf{t} - \mathbf{y})^T (\mathbf{t} - \mathbf{y}) = \frac{1}{2}(\mathbf{t} - \mathbf{X}\mathbf{w})^T (\mathbf{t} - \mathbf{X}\mathbf{w})$$

$$\text{where : } \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \dots \\ t_n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \dots \\ \mathbf{x}_n^T \end{bmatrix}$$

- Minimizing the cost gives the least-square solution

$$\frac{\partial J}{\partial \mathbf{w}} = 0 = \frac{\partial J}{\partial \mathbf{w}}\left(\frac{1}{2}(\mathbf{t} - \mathbf{X}\mathbf{w})^T (\mathbf{t} - \mathbf{X}\mathbf{w})\right) = -(\mathbf{t} - \mathbf{X}\mathbf{w})^T \mathbf{X}$$

$$= -\mathbf{t}^T \mathbf{X} + (\mathbf{X}\mathbf{w})^T \mathbf{X} = -\mathbf{t}^T \mathbf{X} + \mathbf{w}^T \mathbf{X}^T \mathbf{X}$$

$$\textit{thus}: \quad \mathbf{t}^T \mathbf{X} = \mathbf{w}^T \mathbf{X}^T \mathbf{X} \quad \textit{or} \quad \mathbf{X}^T \mathbf{t} = \mathbf{X}^T \mathbf{X} \mathbf{w}$$

$$\textit{result}: \quad \mathbf{w} = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{t}$$

Recursive Least Squares: An Incremental Version of Linear Regression

- Based on the matrix inversion theorem:

$$(\mathbf{A}-\mathbf{BC})^{-1}=\mathbf{A}^{-1}+\mathbf{A}^{-1}\mathbf{B}\left(\mathbf{I}+\mathbf{CA}^{-1}\mathbf{B}\right)^{-1}\mathbf{CA}^{-1}$$

- Incremental updating of a linear regression model

Initialize: $\mathbf{P}^{n}=\mathbf{I}\dfrac{1}{\gamma}$ where $\gamma<<1$ (note $\mathbf{P}\equiv\left(\mathbf{X}^{T}\mathbf{X}\right)^{-1}$

For every new data point $(\mathbf{x},\mathbf{t})$

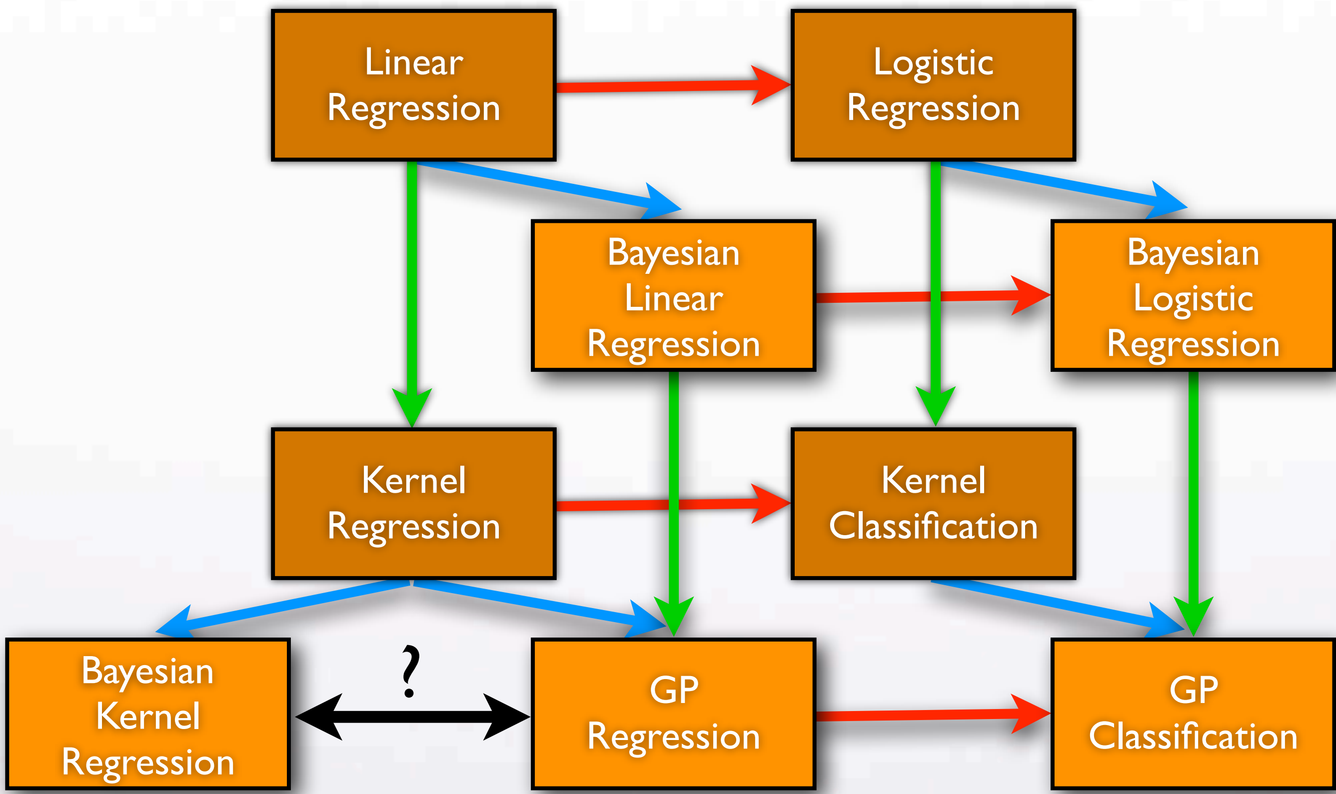
(note that $\mathbf{x}$ includes the bias):

$$\mathbf{P}^{n+1}=\frac{1}{\lambda}\left(\mathbf{P}^{n}-\frac{\mathbf{P}^{n}\mathbf{x}\mathbf{x}^{T}\mathbf{P}^{n}}{\lambda+\mathbf{x}^{T}\mathbf{P}^{n}\mathbf{x}}\right)\text{ where }\lambda=\begin{cases}1 \text{ if no forgetting}\\ <1 \text{ if forgetting}\end{cases}$$

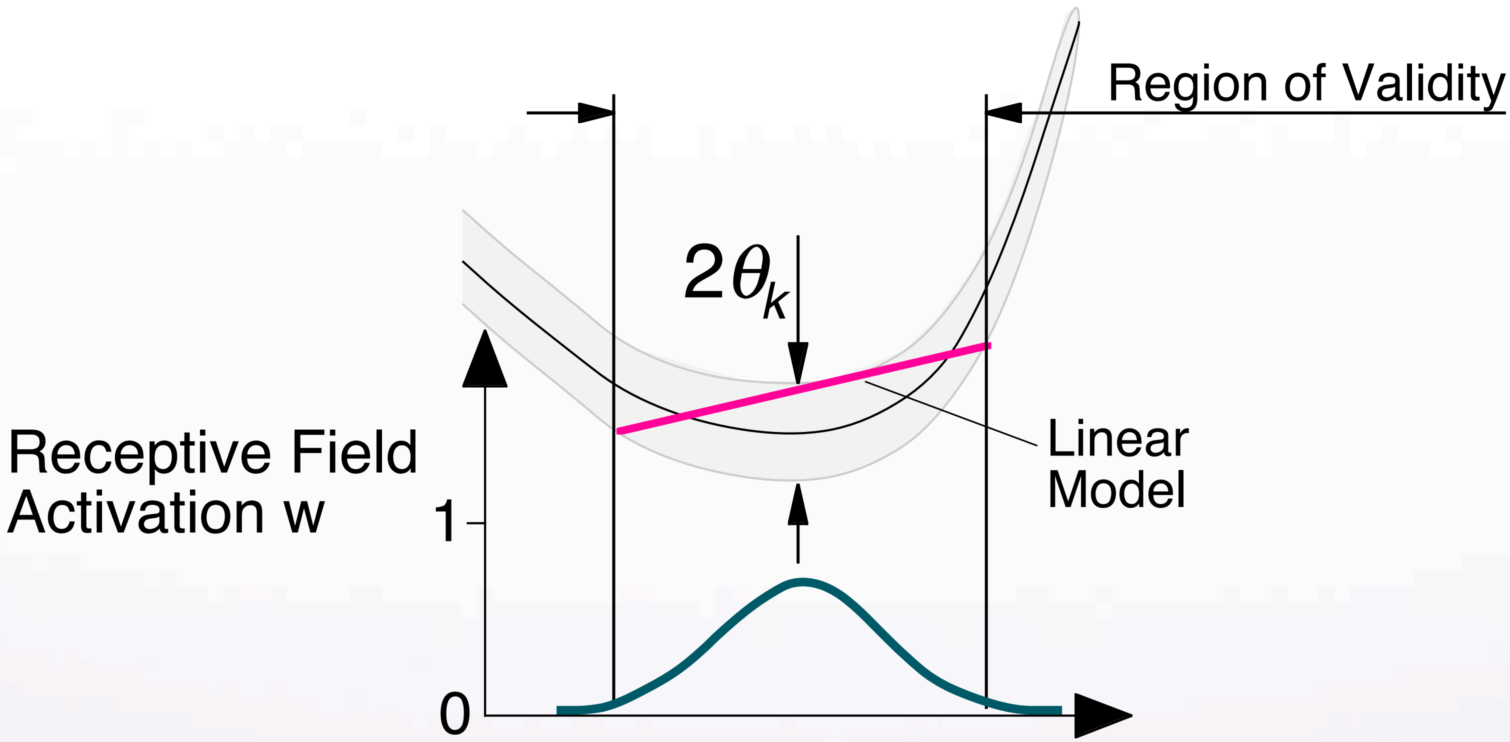
$$\mathbf{W}^{n+1}=\mathbf{W}^{n}+\mathbf{P}^{n+1}\mathbf{x}\left(\mathbf{t}-\mathbf{W}^{nT}\mathbf{x}\right)^{T}$$

- NOTE: RLS gives exactly the same solution as linear regression if no forgetting

Traversing Zoubin's Diagram



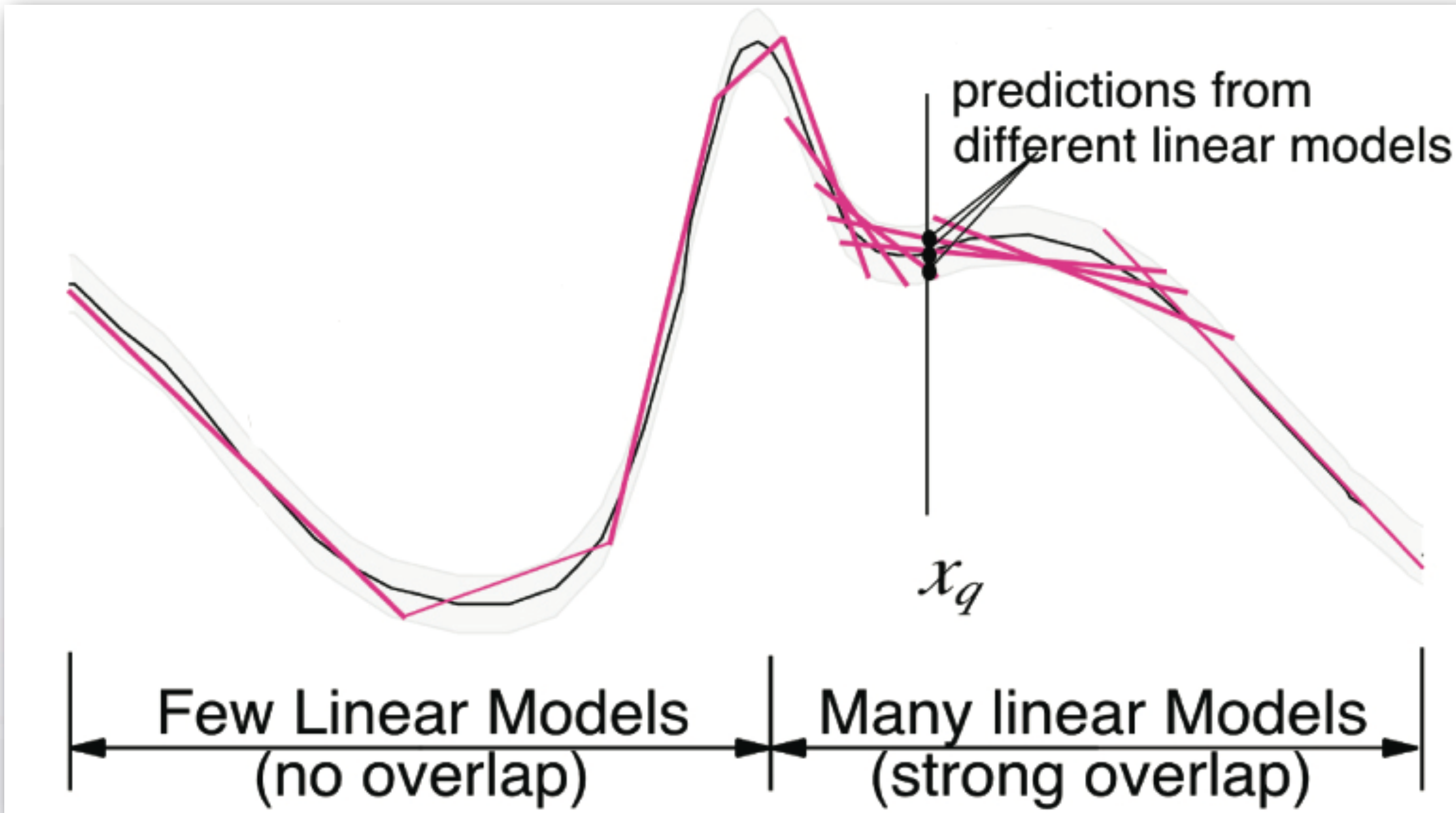
Making Linear Regression Nonlinear: Locally Weighted Regression



$$J = \sum_{i=1}^{N} w_i \left(y_i - \mathbf{x}_i^T \boldsymbol{\beta} \right)^2$$

Note: Using GPs, SVR, Mixture Models, etc., are other ways to nonlinear regression

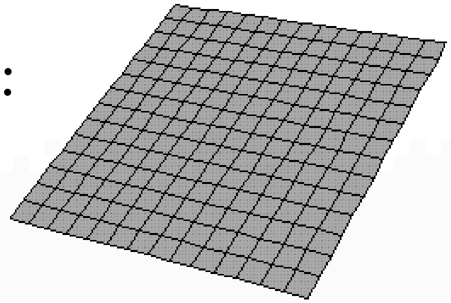
Locally Weighted Regression



- Piecewise linear function approximation,
- Each local model is learned from only local data
- No over-fitting due to too many local models (unlike RBFs, ME)

Locally Weighted Regression

Linear Model:



learned with

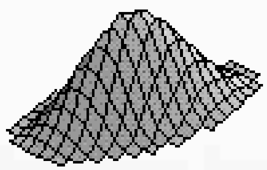
Recursive weighted least squares:

$$\beta_k^{n+1} = \beta_k^n + w\mathbf{P}_k^{n+1}\mathbf{x}\left(\mathbf{y} - \tilde{\mathbf{x}}^T\beta_k^n\right)^T$$

$$\mathbf{P}_k^{n+1} = \frac{1}{\lambda}\left(\mathbf{P}_k^n - \frac{\mathbf{P}_k^n\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\mathbf{P}_k^n}{\frac{\lambda}{w} + \tilde{\mathbf{x}}^T\mathbf{P}_k^n\tilde{\mathbf{x}}}\right)$$

$$\mathbf{y} = \beta_x{}^T\mathbf{x} + \beta_0 = \beta^T\tilde{\mathbf{x}} \quad \text{where} \quad \tilde{\mathbf{x}} = \left[\mathbf{x}^T\ 1\right]^T$$

Weighting Kernel:



learned with

Gradient descent in penalized leave-one-out local cross-validation (PRESS) cost function:

$$\mathbf{M}_k^{n+1} = \mathbf{M}_k^n - \alpha\frac{\partial J}{\partial \mathbf{M}}$$

$$J = \frac{1}{\sum_{i=1}^{N} w_{k,i}} \sum_{i=1}^{N} w_{k,i}\left\|\mathbf{y}_i - \hat{\mathbf{y}}_{k,i,-i}\right\|^2 + \gamma \sum_{i=1,j=1}^{n} D_{k,ij}^2$$

$$w = \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{c})^T\mathbf{D}(\mathbf{x}-\mathbf{c})\right) \quad \text{where} \quad \mathbf{D} = \mathbf{M}^T\mathbf{M}$$

Combined Prediction:

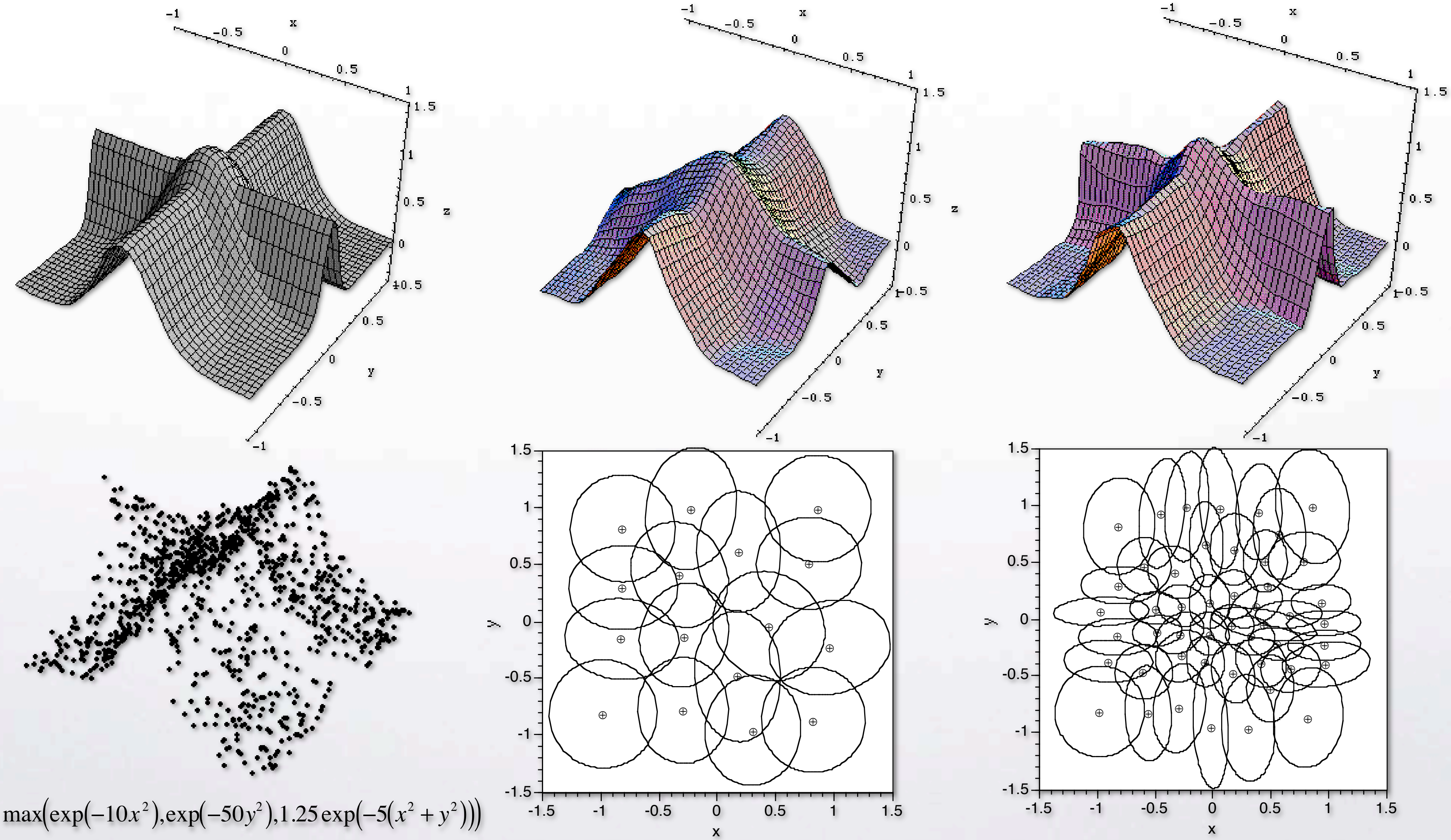
$$\mathbf{y} = \frac{\sum_{i=1}^{K} w_i \mathbf{y}_k}{\sum_{i=1}^{K} w_i}$$

add model when

$$\mathit{if} \quad \min_k\left(w_k\right) < w_{gen}$$

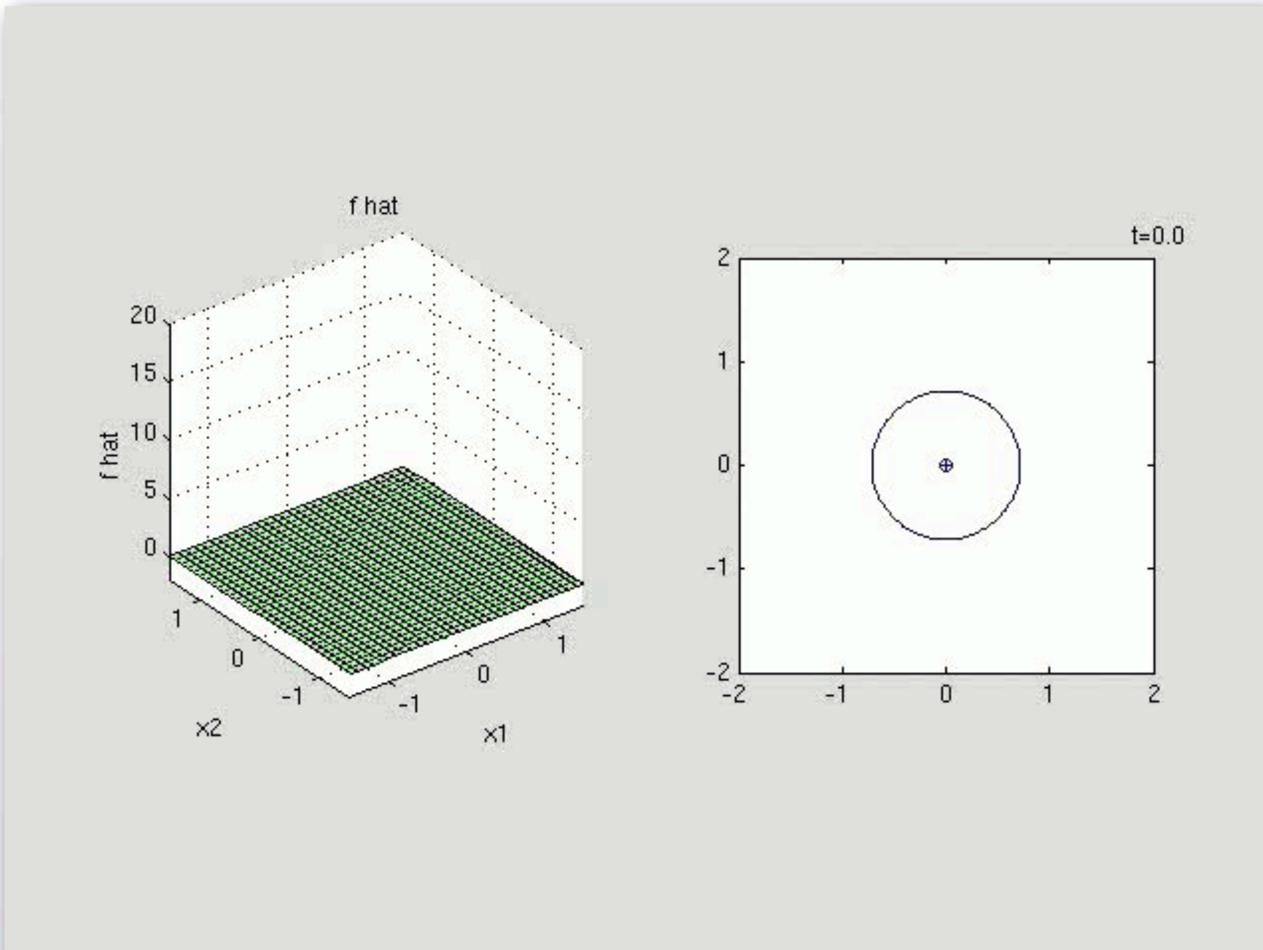
$$\mathit{create\,new\,RF\,at} \quad \mathbf{c}_{K+1} = \mathbf{x}$$

Locally Weighted Regression

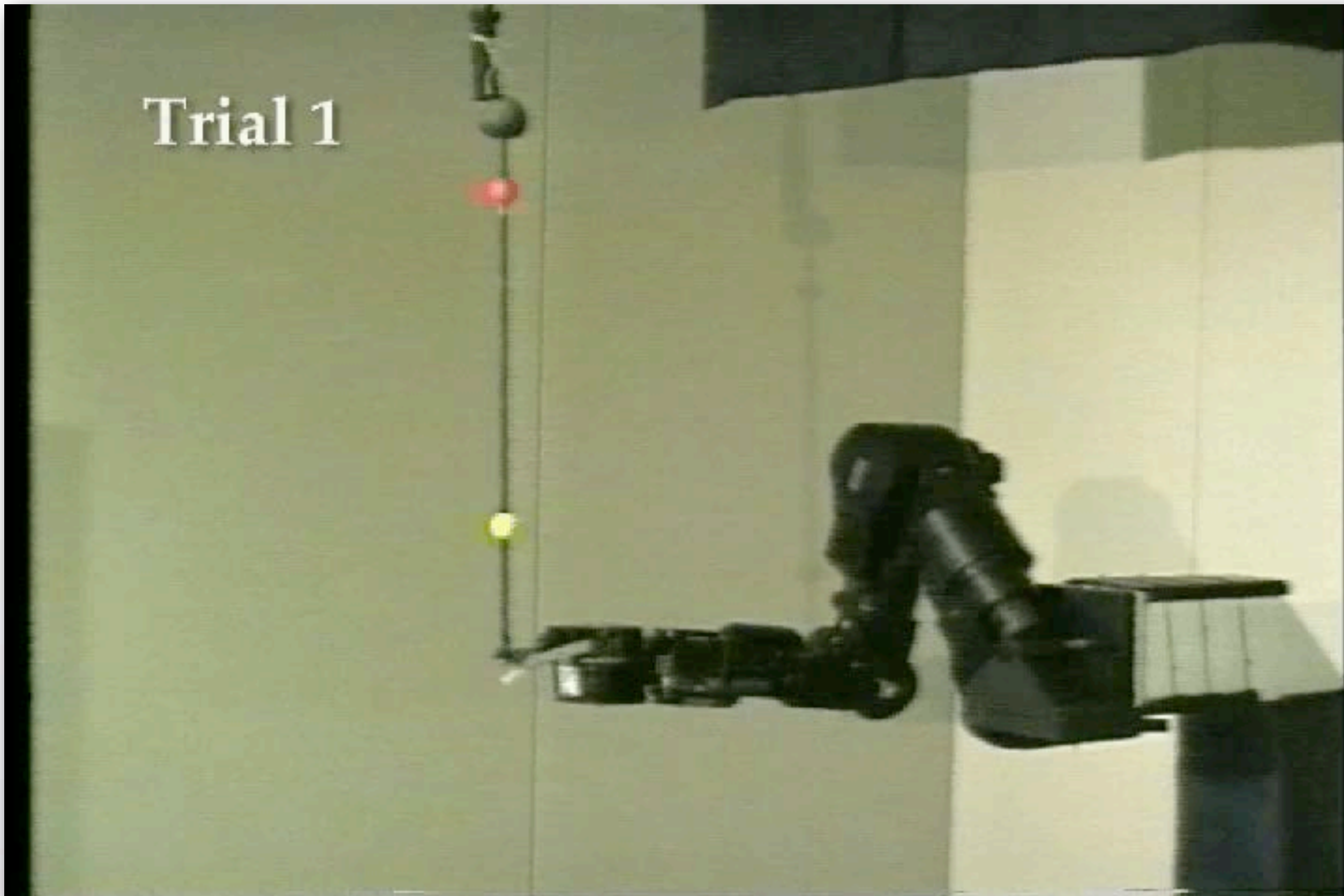


$$z = \max\left(\exp(-10x^2), \exp(-50y^2), 1.25\exp(-5(x^2+y^2))\right)$$

Locally Weighted Regression Inserted into Adaptive Control

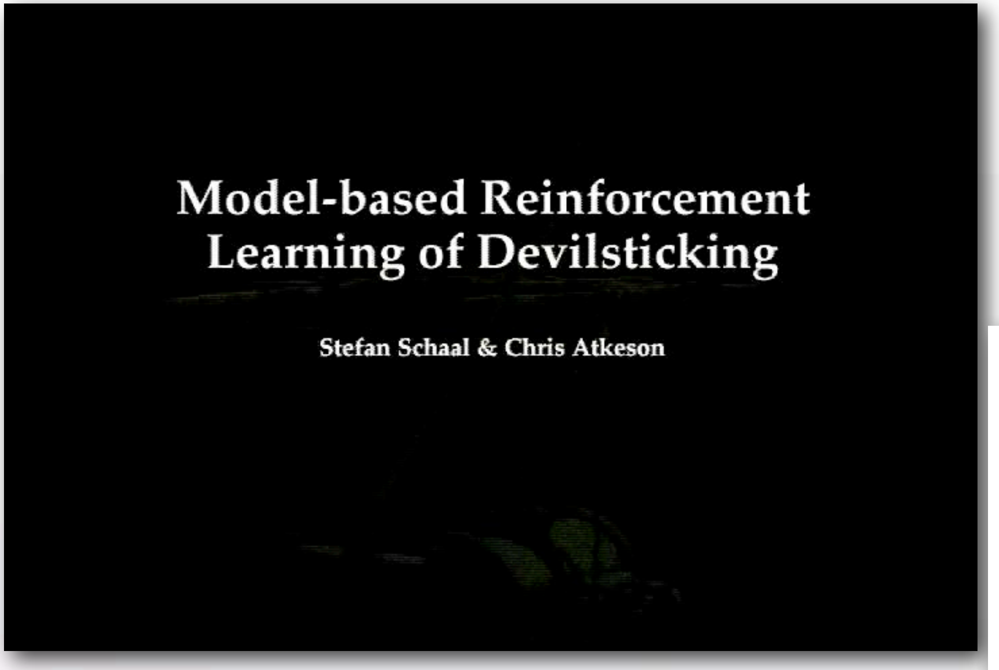


Locally Weighted Regression



Learn forward model of task dynamics,
then computer controller

Locally Weighted Regression



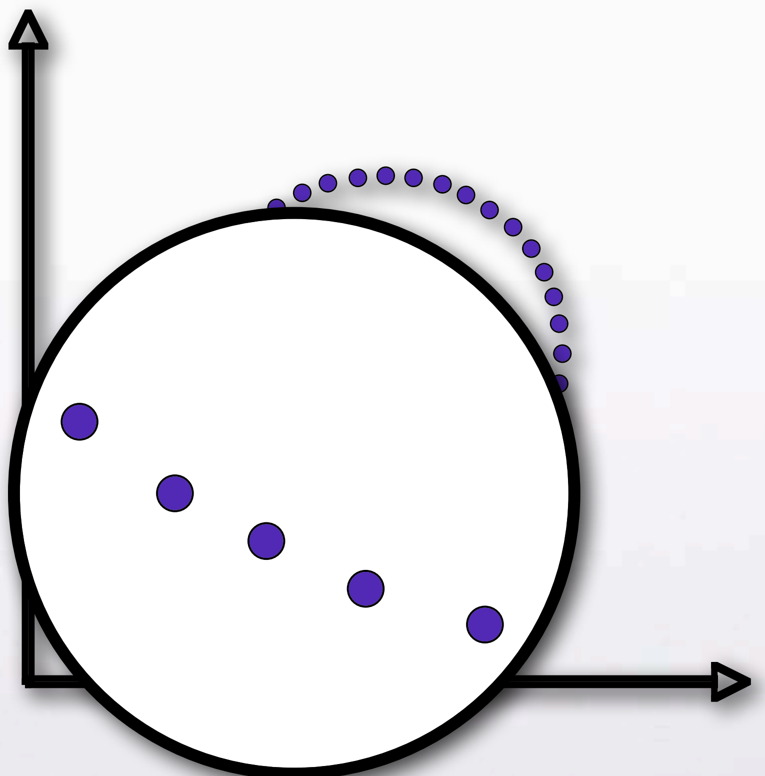
Learn forward model of task dynamics,
then computer controller

Criticism of Locally Weighted Learning

- Breaks down in high-dimensional spaces
- Computationally expensive and numerically brittle due to (incremental) dxd matrix inversion
- Not compatible with modern probabilistic statistical learning algorithms
- Too many “manual tuning parameters”

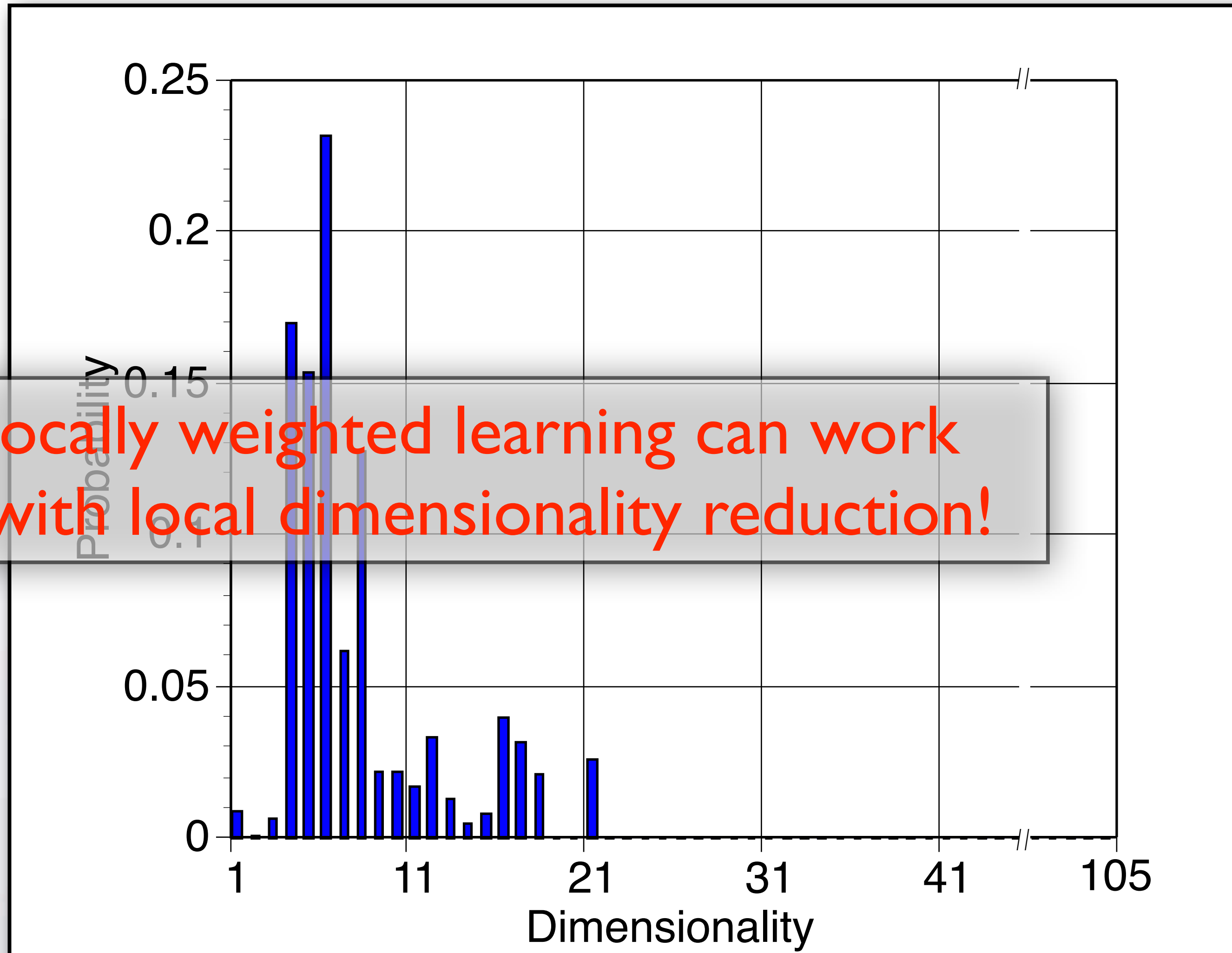
The Curse of Dimensionality

- The power of local learning comes from exploiting the discriminative power of local neighborhood relations.
- But the notion of a “local” breaks down in high dim. spaces!



The Curse of Dimensionality

Movement Data is Locally Low Dimensional

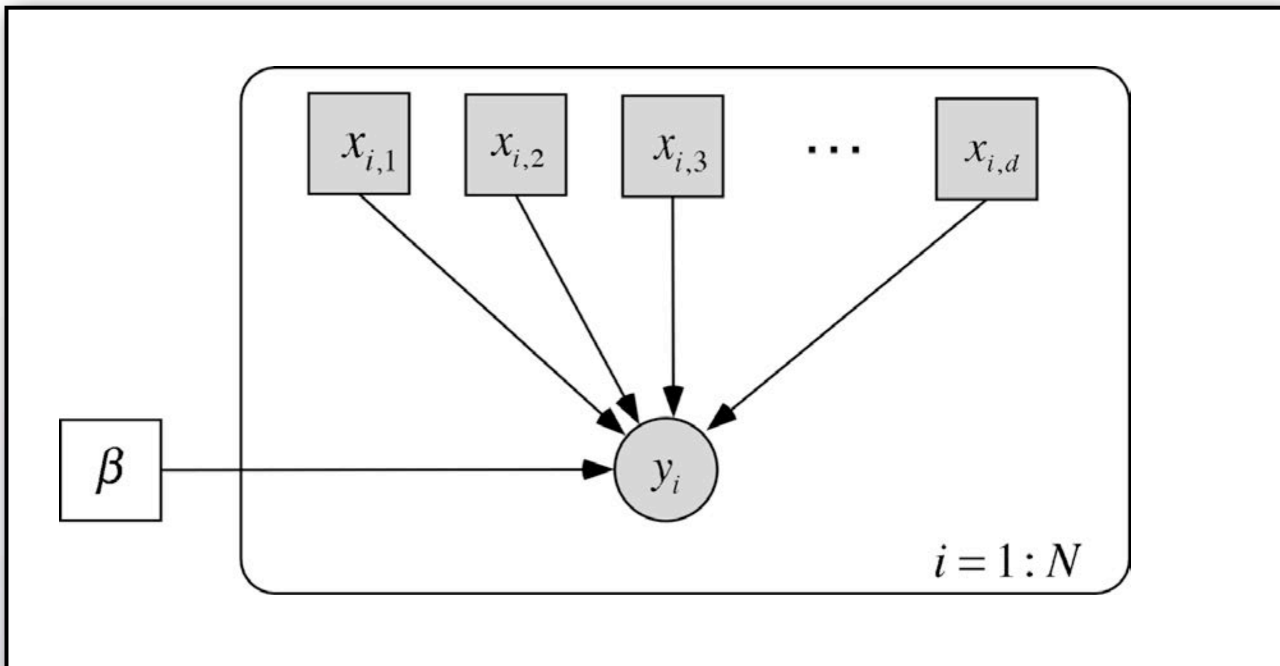


Thus, locally weighted learning can work
if used with local dimensionality reduction!

Derived with Bayesian Factor Analysis

A Bayesian Approach to Locally Weighted Learning

- Linear Regression as a Graphical Model



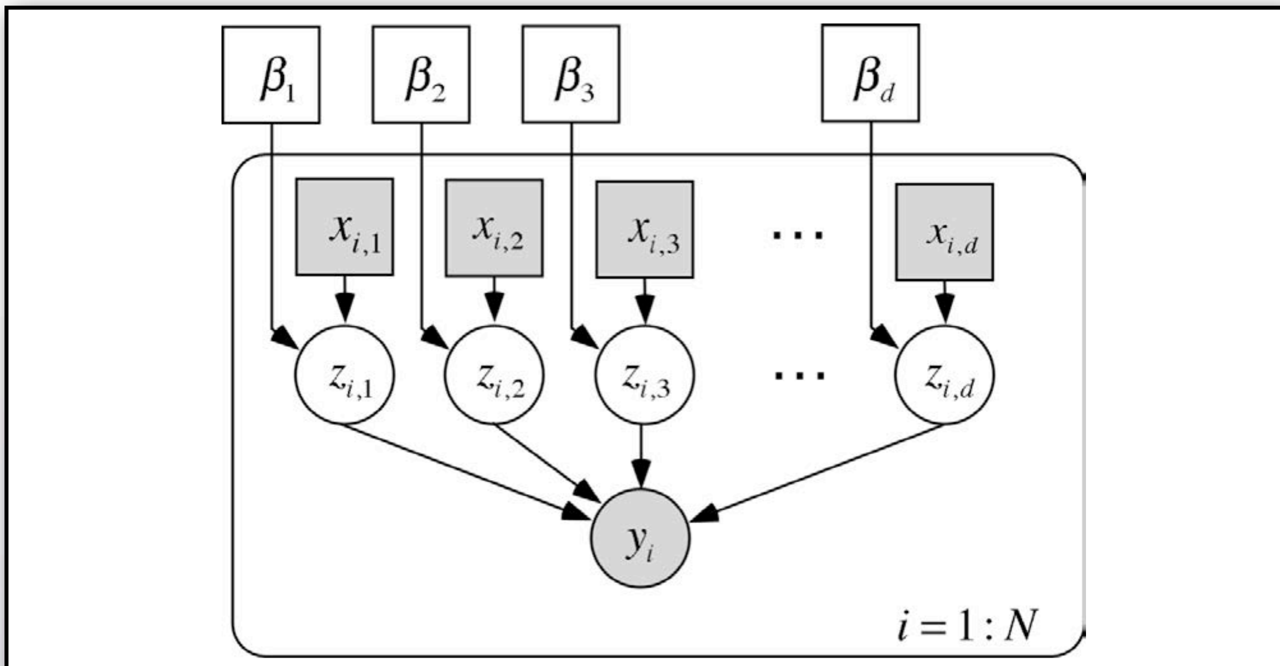
$$y_i = \mathbf{x}_i^T \beta + \varepsilon$$

$$\varepsilon \sim N\left(0, \psi_y\right)$$

$$\beta = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}\mathbf{y}$$

A Bayesian Approach to Locally Weighted Learning

- Inserting a Partial-Least-Squares-like projection as a set of hidden variables



$$z_{i,m} = x_{i,m}\beta_j + \eta_m$$

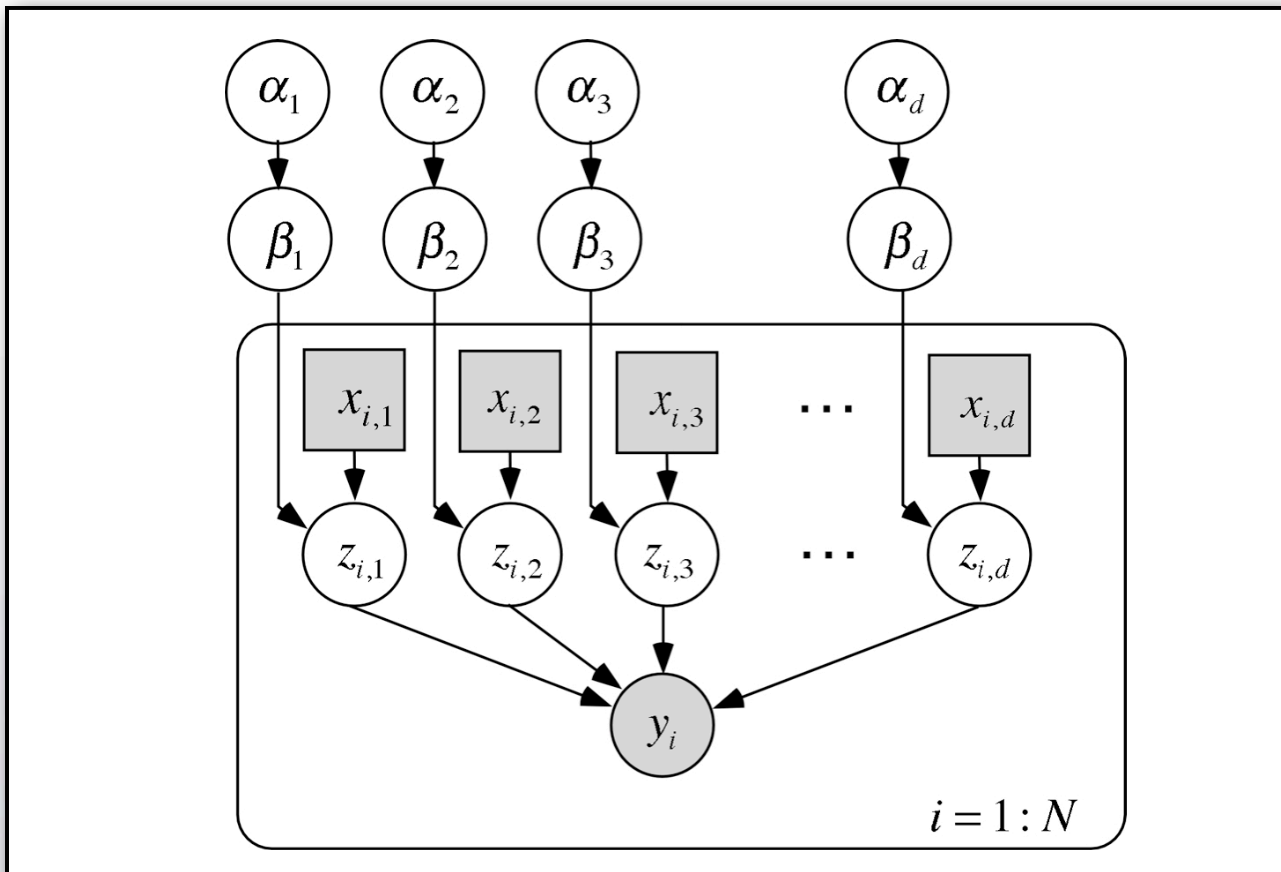
$$y_i = \sum_{m=1}^{d} z_{i,m} + \varepsilon$$

$$\varepsilon \sim N\left(0, \psi_y\right)$$

$$\eta_m \sim N\left(0, \psi_{z,m}\right)$$

A Bayesian Approach to Locally Weighted Learning

- Robust linear regression with automatic relevance detection (ARD, sparsification)



$$z_{i,m} = x_{i,m}\beta_j + \eta_m$$

$$y_i = \sum_{m=1}^{d} z_{i,m} + \varepsilon$$

$$\varepsilon \sim N\left(0, \psi_y\right)$$

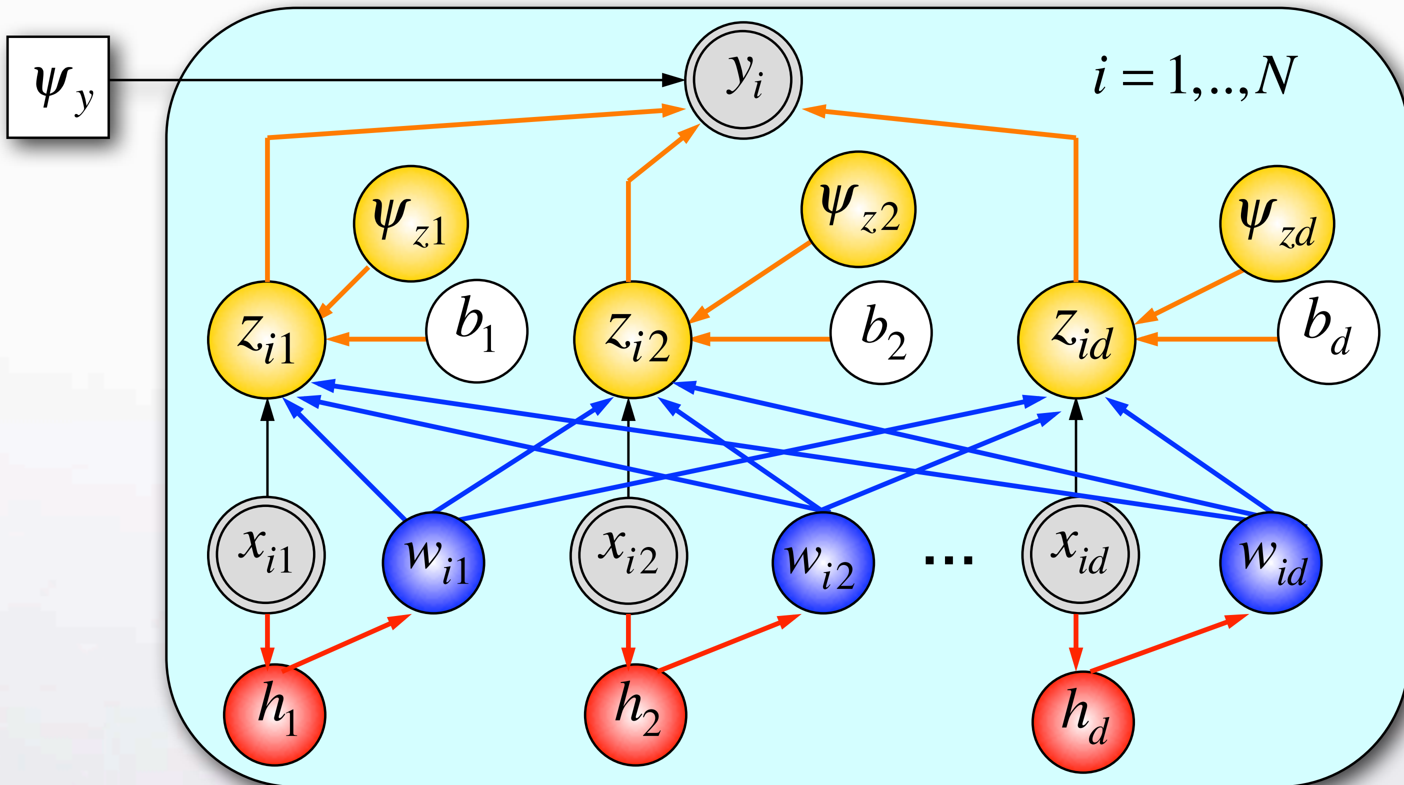
$$\eta_m \sim N\left(0, \psi_{z,m}\right)$$

$$\beta_m \sim N\left(0, \frac{1}{\alpha_m}\right)$$

$$\alpha_m \sim Gamma\left(a_\alpha, b_\alpha,\right)$$

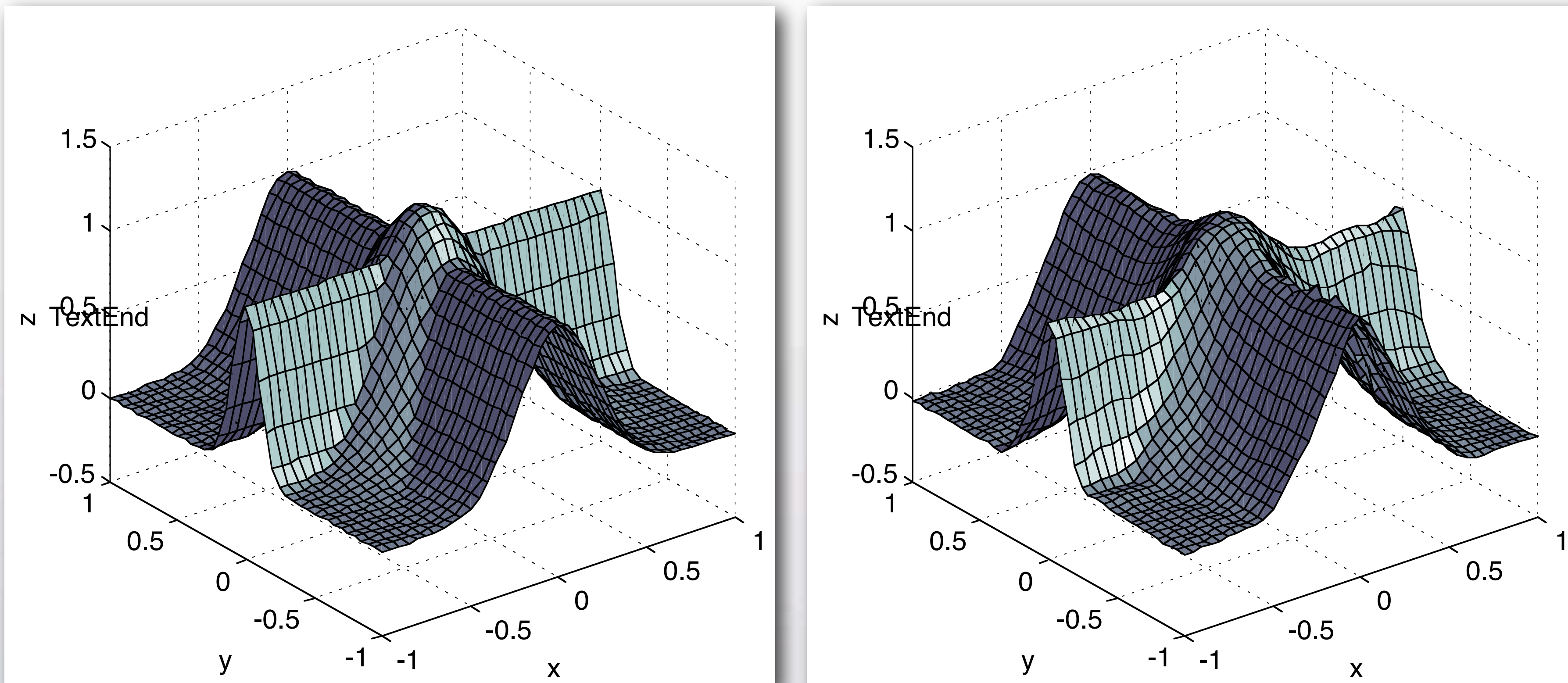
A Full Bayesian Treatment of Locally Weighted Learning

- The final model for full Bayesian parameter adaptation for regression and locality



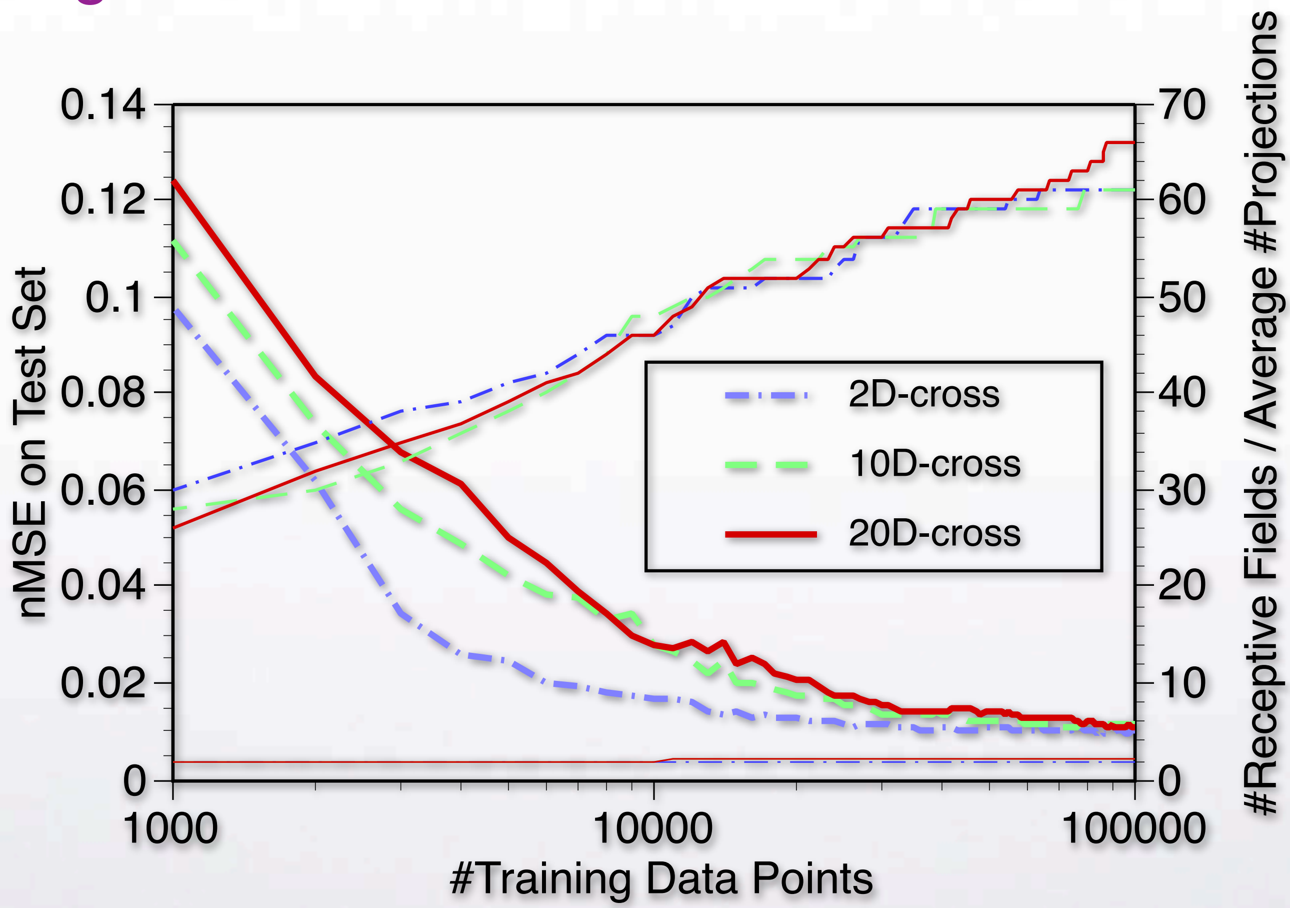
Locally Weighted Learning In High Dimensional Spaces

- Learning the “cross” function in 20-dimensional space



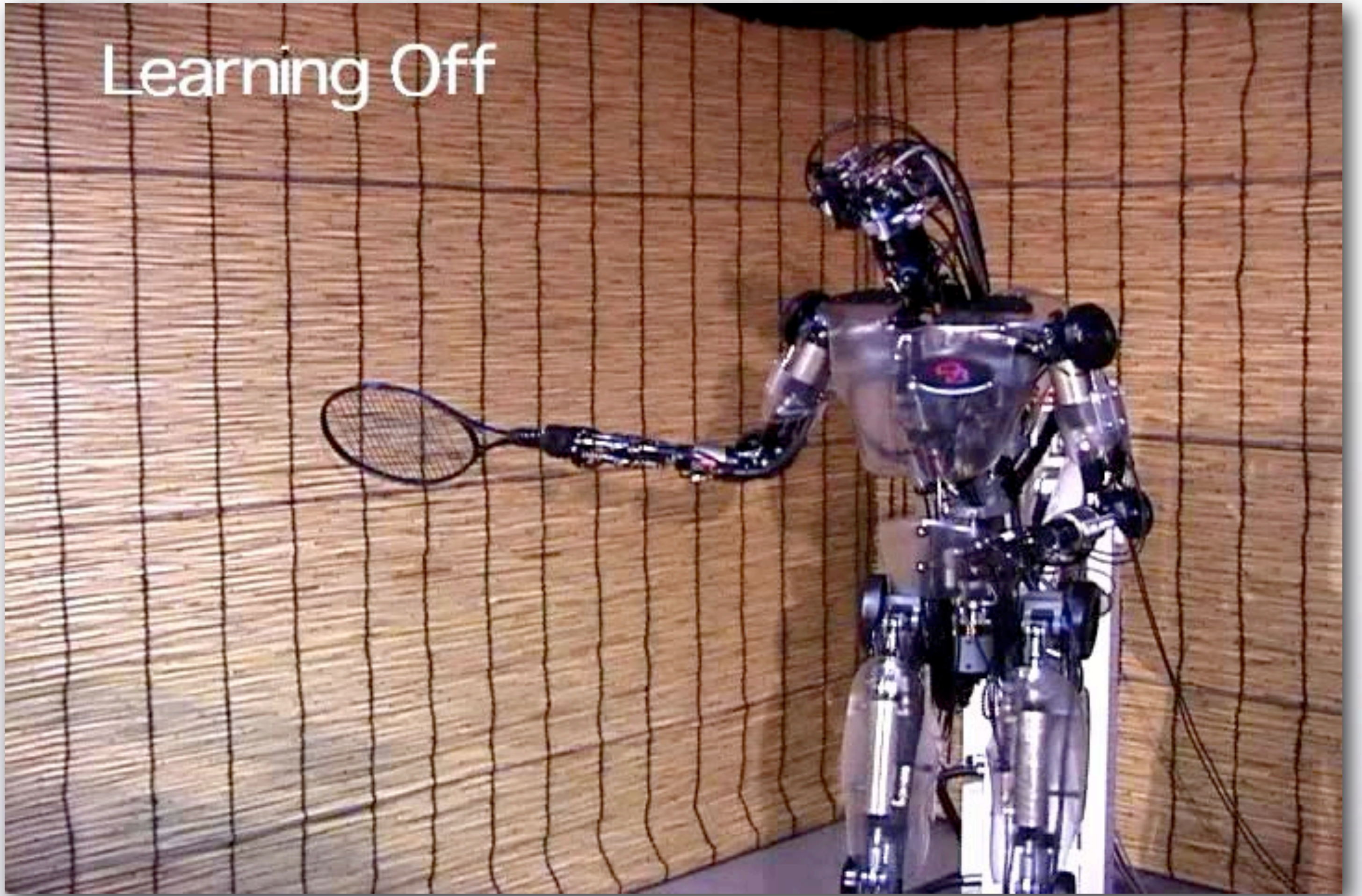
Locally Weighted Learning In High Dimensional Spaces

- Learning the “cross” function in 20-dimensional space



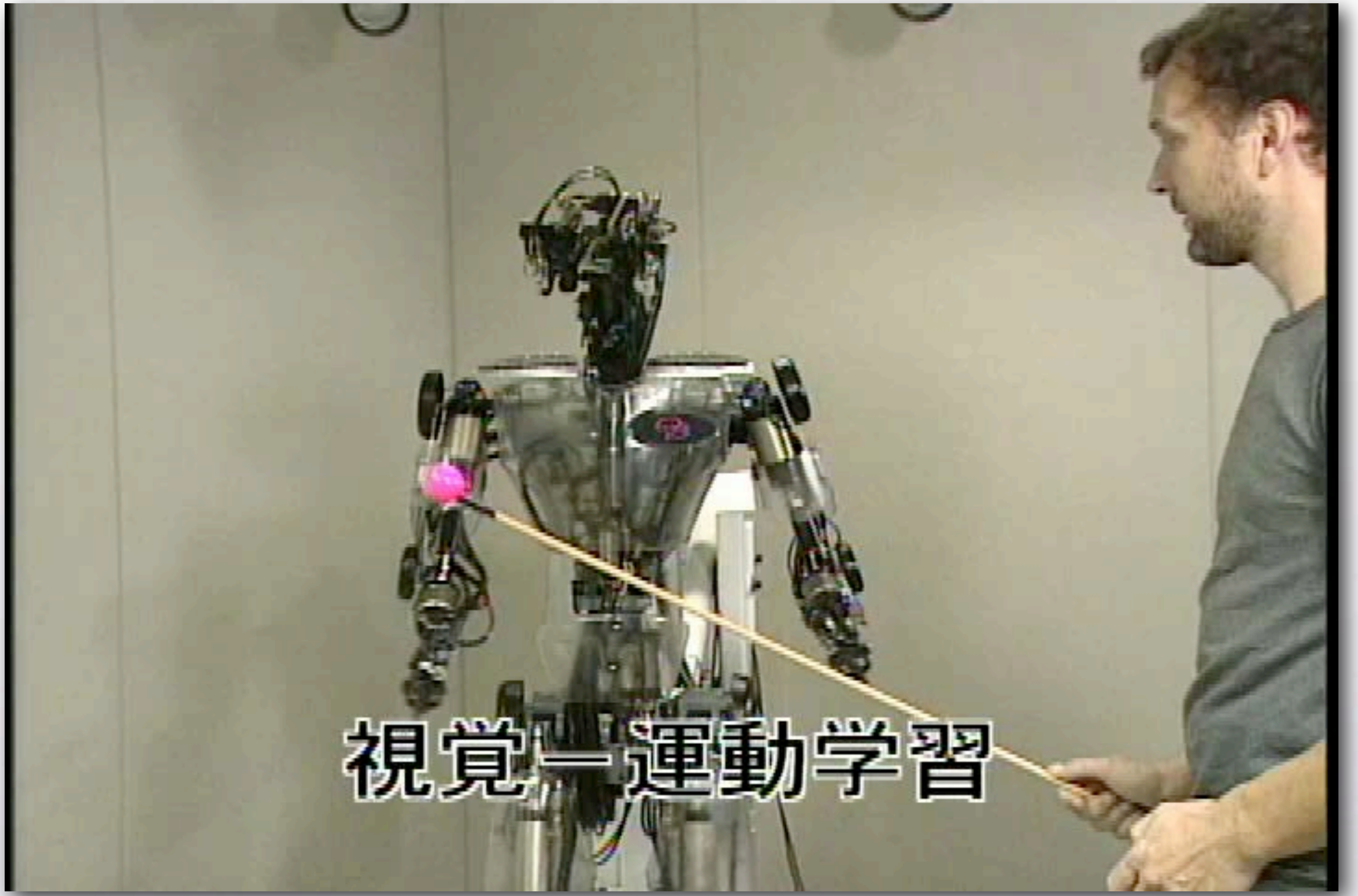
Locally Weighted Learning In High Dimensional Spaces

- Learning internal models in 90 dimensional space



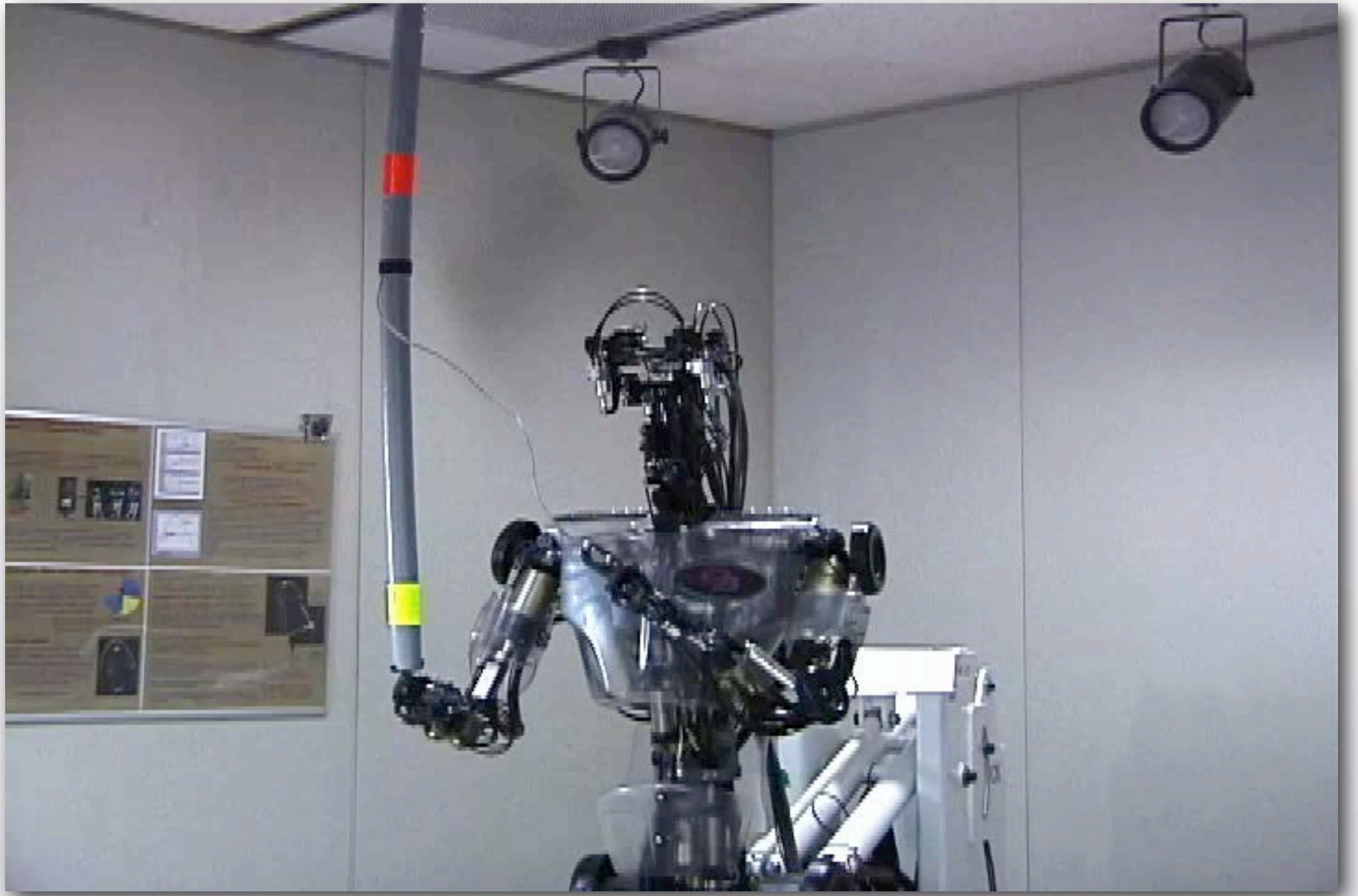
Locally Weighted Learning In High Dimensional Spaces

- Learning inverse kinematics in 60 dimensional space



Locally Weighted Learning In High Dimensional Spaces

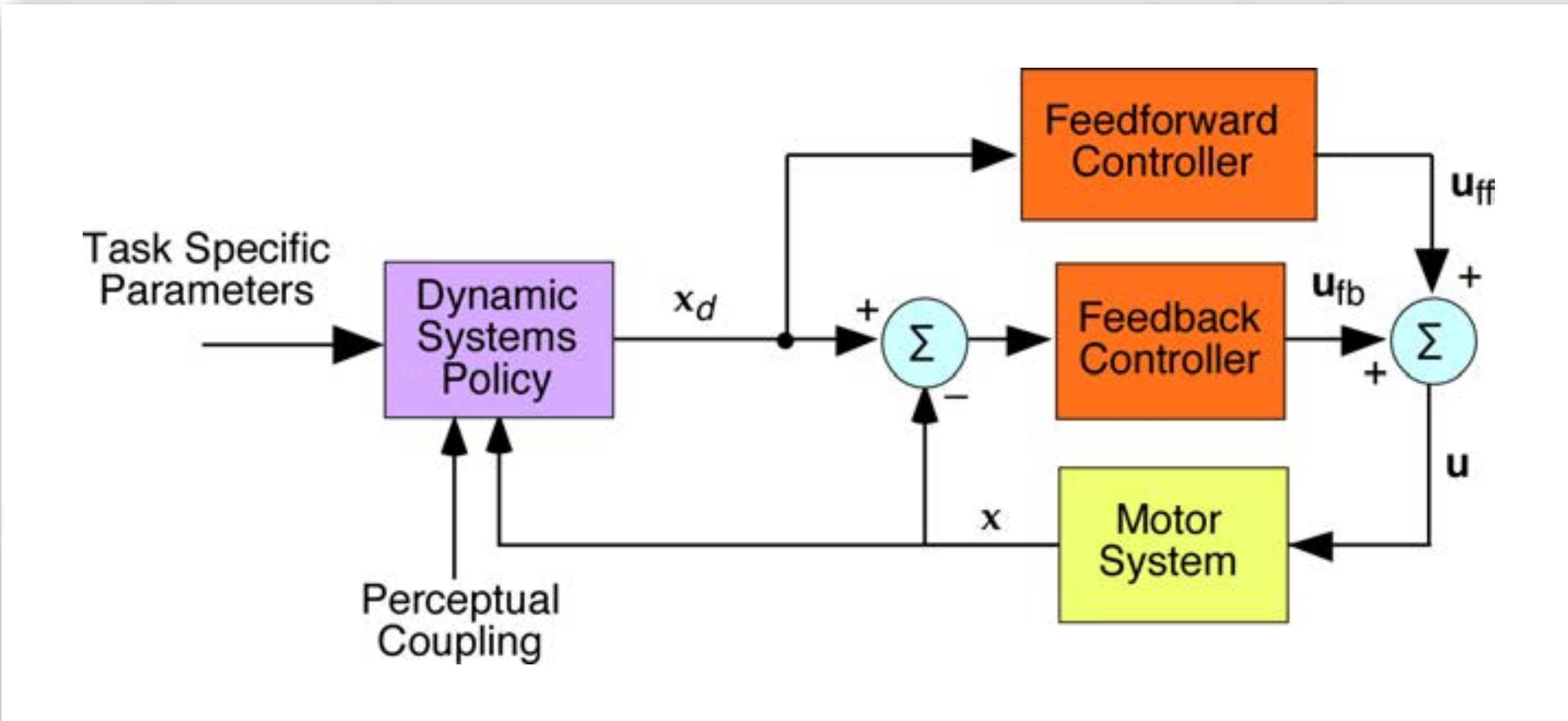
- Skill learning



Outline

- A Bit of Robotics History
- Foundations of Control
- Adaptive Control
- Learning Control
    - Model-based Robot Learning
    - Reinforcement Learning

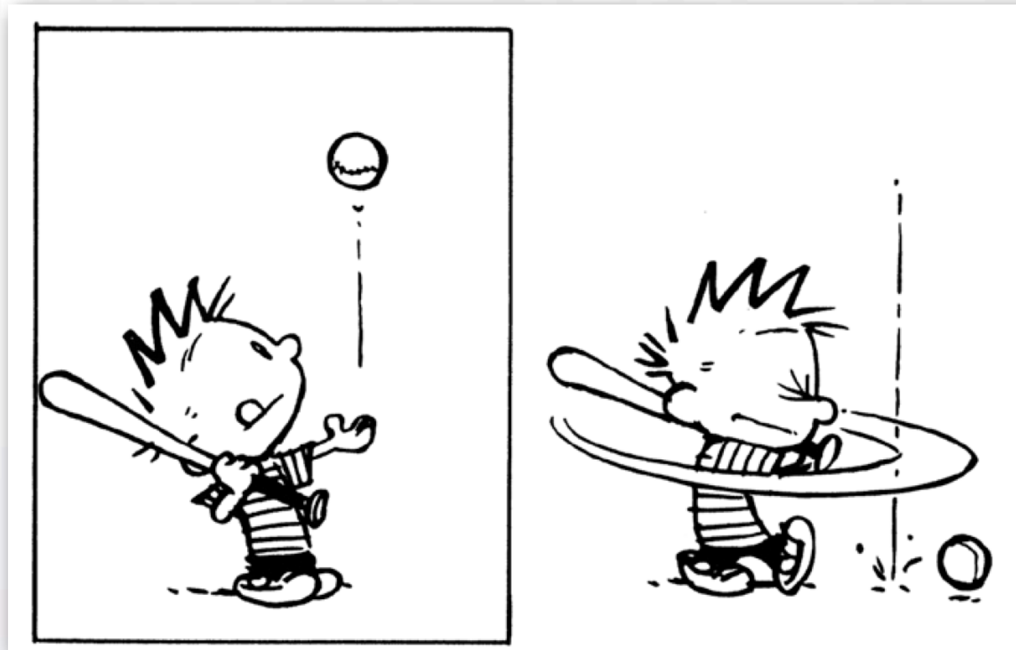
Given: A Parameterized Policy and a Controller



Note: we are now starting to address planning, i.e,. where do desired trajectories come from?

Trial & Error Learning
Reinforcement Learning from Trajectories

- Problem:
  - How can a motor system learn a novel motor skill?
  - Reinforcement learning is a general approach to this problem, but little work has been done to scale to the high-dimensional continuous state-action domains of humans
- Approach:
  - Teach with imitation learning the initial skill using a parameterized control policy
  - Provide an objective function for the skill
  - Perform trial-and-error learning from exploratory trajectories



Reinforcement Learning Terminology

- Policies
  - perceived state to action mapping (can be probabilistic)
- Reward functions
  - maps the perceived state-action pair into a a single number, an immediate reward (stochastic)
- Value functions
  - maps the state into the accumulated expected reward that would be received if starting in the state
- Models
  - predicts the next state given the current state and action (can be probabilistic)

- **Policy**: what to do
- **Reward**: what is good
- **Value**: what is good because it *predicts* reward
- **Model**: what follows what

Objective: Optimize Reward!

Value Functions

- The value of a state is the expected return starting from that state; depends on the agent's policy:

State - value function for policy π :

$$V^{\pi}(\mathbf{x}) = E_{\pi}\left\{R_t \mid \mathbf{x}_t = \mathbf{x}\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid \mathbf{x}_t = \mathbf{x}\right\}$$

- The value of taking an action in a state under policy π is the expected return starting from that state, taking that action, and thereafter following π :

Action - value function for policy π :

$$Q^{\pi}(\mathbf{x},\mathbf{u}) = E_{\pi}\left\{R_t \mid \mathbf{x}_t = \mathbf{x}, \mathbf{u}_t = \mathbf{u}\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid \mathbf{x}_t = \mathbf{x}, \mathbf{u}_t = \mathbf{u}\right\}$$

Bellman Equation for a Policy π

The basic idea:

$$
\begin{aligned}
R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots \\
&= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \cdots \right) \\
&= r_{t+1} + \gamma R_{t+1}
\end{aligned}
$$

So:

$$
\begin{aligned}
V^{\pi}(\mathbf{x}) &= E_{\pi} \left\{ R_t \,\middle|\, \mathbf{x}_t = \mathbf{x} \right\} \\
&= E_{\pi} \left\{ r_{t+1} + \gamma V\left(\mathbf{x}_{t+1}\right) \,\middle|\, \mathbf{x}_t = \mathbf{x} \right\}
\end{aligned}
$$

Bellman Optimality Equation for V*

- The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$
\begin{aligned}
V^*(\mathbf{x}) &= \max_{\mathbf{u}\in A(\mathbf{x})} Q^{\pi}(\mathbf{x},\mathbf{u}) \\
&= \max_{\mathbf{u}\in A(\mathbf{x})} E\left\{ r_{t+1} + \gamma V^*\left(\mathbf{x}_{t+1} \mid \mathbf{x}_t = \mathbf{x}, \mathbf{u}_{\mathbf{t}} = \mathbf{u}\right)\right\}
\end{aligned}
$$

V^* is the unique solution of this system of equations.

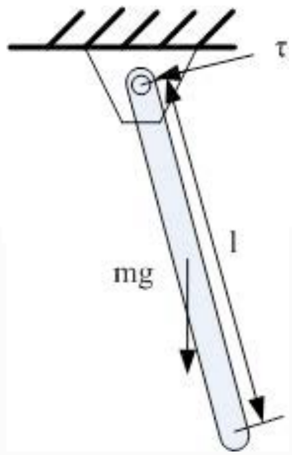
Bellman Optimality Equation for Q*

- The value of a state/action under an optimal policy must equal the expected return for this action from that state, and then following the optimal policy:

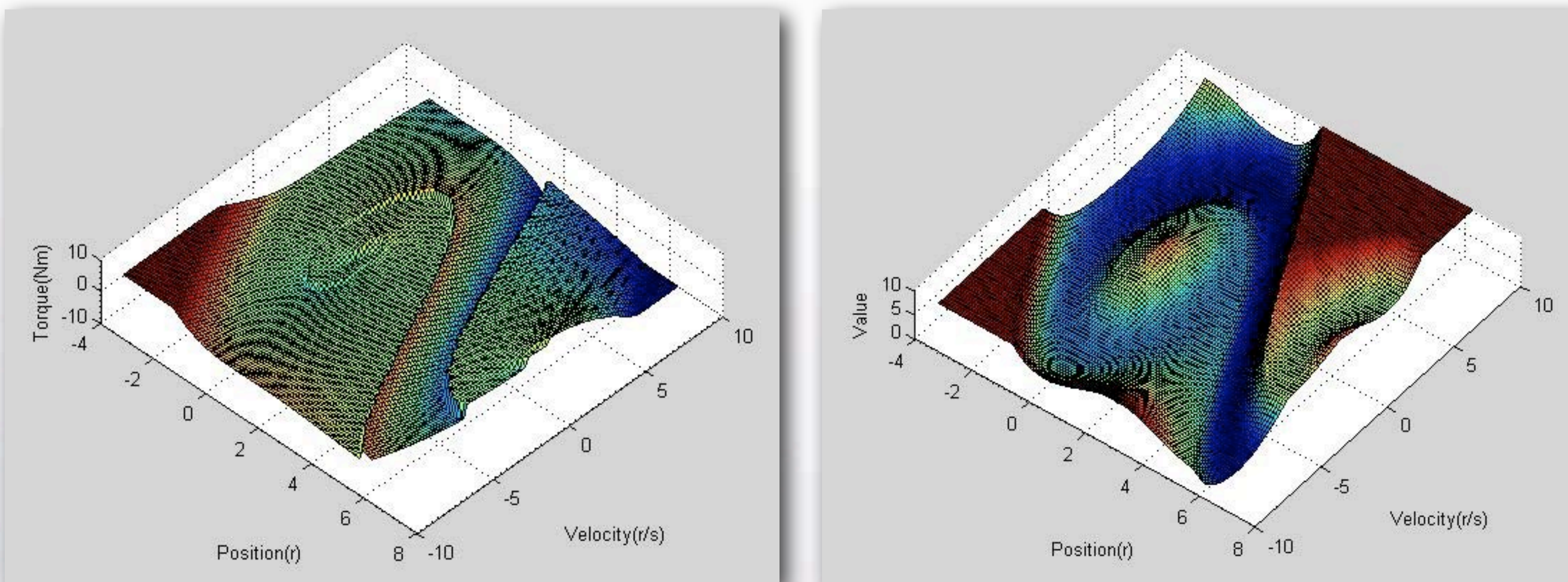
$$Q^*(\mathbf{x},\mathbf{u}) = E\left\{ r_{t+1} + \gamma \max_{\mathbf{u}'} Q^*(\mathbf{x}_{t+1},\mathbf{u}') \middle| \mathbf{x}_t = \mathbf{x}, \mathbf{u}_t = \mathbf{u} \right\}$$

Q^* is the unique solution of this system of equations.

Example: Learning a Pendulum Swing-Up



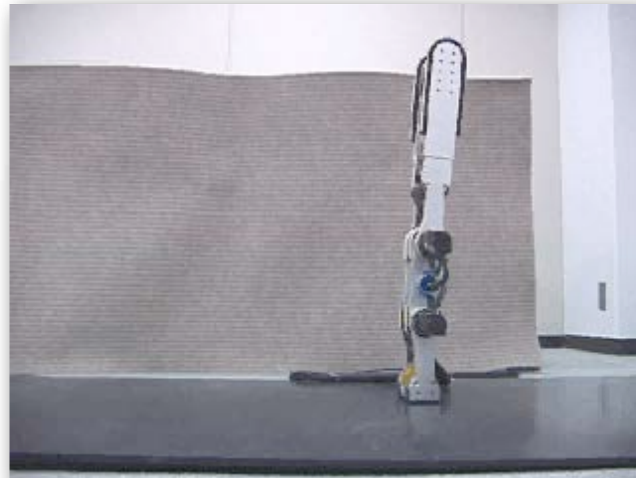
Note: Both policy and value function are rather complex landscapes with discontinuities!



Policy

Value Function

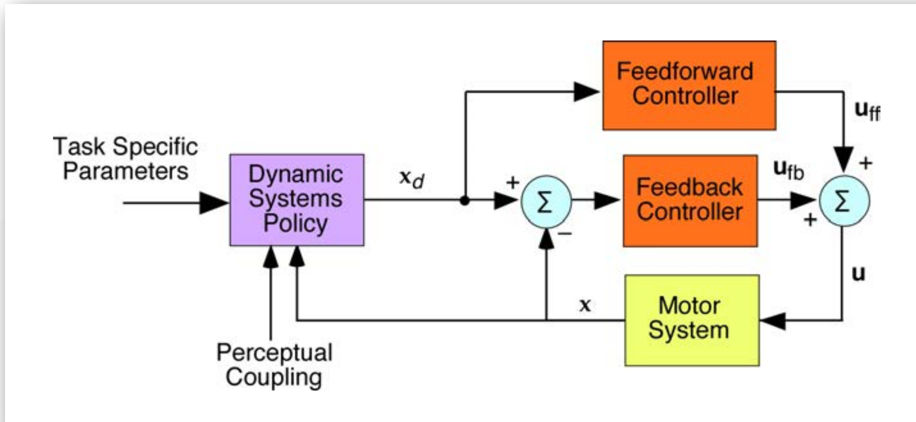
Some More Exciting Examples



State-Based vs. Trajectory-based Reinforcement Learning

- From about 1980-2000, value function-based (i.e., state-based) reinforcement learning has been dominant (textbook Sutton&Barto)
  - Pros:
    - well-understood theory
    - convergence proofs for discrete state-action systems
    - a useful set of algorithms to work with (model-based and model-free)
    - ideally a globally optimal solution
  - Cons:
    - problematic in continuous state-action spaces (max-operator in continuous spaces)
    - curse of dimensionality in high-dimensional systems
    - hard to combine with function approximation
    - greed (= agressive) updating
- Trajectory-based reinforcement learning
  - Pros:
    - can work in high dimensional continuous state-action spaces
    - does not suffer from the curse of dimensionality
  - Cons:
    - Locally optimal solutions
    - classical methods learn very slowly

Trajectory-based Reinforement Learning with Parameterized Policies



$$\mathbf{u}(t) = \pi(\mathbf{x}(t), t, \alpha)$$

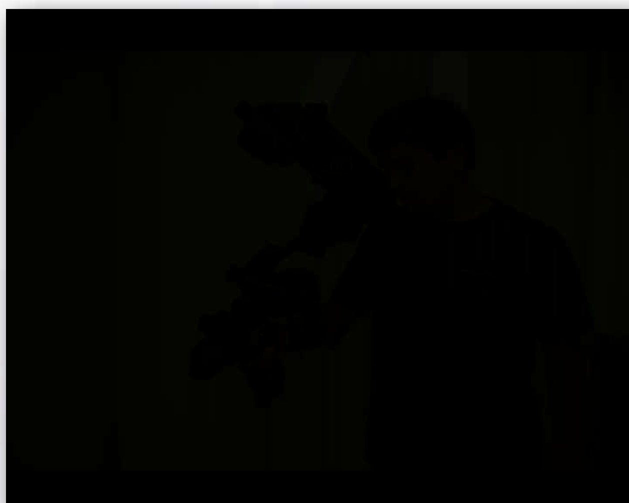
or

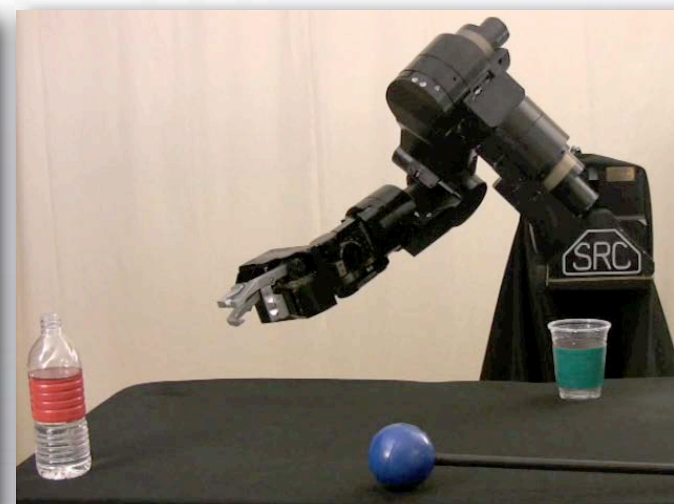
$$\dot{\mathbf{x}}_d(t) = \pi(\mathbf{x}_d(t), t, \alpha)$$

Example: Dynamic Systems Policies, initalized by imitation

$$\tau \ddot{y} = \alpha_z(\beta_z(g-y) - \dot{y}) + \frac{\sum_{i=1}^{k} w_i b_i x}{\sum_{i=1}^{k} w_i}$$

$$\tau \dot{x} = -\alpha_x x$$





Trajectory-based Reinforcement Learning

- Define a cost function <u>along the trajectory</u>:

$$J = E_{\tau}\left\{\sum_{i=0}^{T} r_i\right\}$$

- And a parameterized control policy (e.g., a movement primitive)

$$\tau\,\dot{\mathbf{y}} = f\left(\mathbf{y}, goal, \mathbf{b}\right)$$

- Optimize *J* with respect to parameters **b**, e.g., by gradient descent

$$\mathbf{b}^{n+1} = \mathbf{b}^{n} + \alpha \frac{\partial J}{\partial \mathbf{b}}$$

Example: Learning with Natural Gradients

Goal: Hit ball to fly far

Note: about 150-200 trials are needed.

Reinforcement Learning from Trajectories

- State-of-the-art of Reinforcement Learning from Trajectories:
  - Given the cost per trajectory τ : $$J = E_\tau \left\{ \sum_{i=0}^{T} r_i \right\}$$
  - The motor primitives with parameters **b**: $$\tau \dot{\mathbf{y}} = f\left(\mathbf{y}, goal, \mathbf{b}\right)$$
  - RL with Natural Gradients $$\mathbf{b}^{new} = \mathbf{b}^{old} + \alpha \frac{\partial J_{NAC}}{\partial \mathbf{b}}$$
  - Probabilistic RL with Reward-Weighted Regression $$\mathbf{b}^{new} \propto \sum_T R_\tau \mathbf{b}_\tau \, / \sum_T R_\tau$$
  - Trajectory-based Q-learning (fitted Q-iteration)
    - an actor-critic based method based on an action-value function over trajectories
  - RL with path-integrals (a probabilistic, model-based/model-free approach derived from stochastic optimal control)

Reinforcement Learning Based on Path Integrals

- Pre-requisites

System Dynamics (Control-Affine):

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},t) + \mathbf{G}(\mathbf{x})\big(\mathbf{u}(t) + \boldsymbol{\varepsilon}(t)\big) = \mathbf{F}(\mathbf{x},\mathbf{u},t)$$

Cost Function:

$$r_t = q(\mathbf{x}_t) + \frac{1}{2}\mathbf{u}_t^T \mathbf{R} \mathbf{u}_t$$

$$J_{\mathbf{x}_t} = E_{\mathbf{x}_t}\left\{ q_T + \int_{t'=t}^{T} r_{t'}\, dt' \right\}$$

Note: this is a more structured approach to RL

$\rightarrow$ Goal: find commands $\mathbf{u}$ that minimize this cost

Reinforcement Learning Based on Path Integrals

- Sketch of the Path-Integral Derivation

Stochastic HJB Equations:

$$-\partial_t V(\mathbf{x}_t,t) = \min_{\mathbf{u}_{t:t_m}} \left[r_t + \partial_{\mathbf{x}} V(\mathbf{x}_t,t)^T \mathbf{F}(\mathbf{x},\mathbf{u},t) + \frac{1}{2} Tr\left\{\Omega(\mathbf{x},\mathbf{u},t)\partial_{\mathbf{x}}^2 V(\mathbf{x}_t,t)\right\}\right]$$

$$\min_{\mathbf{u}_{t:t_m}} \left[\frac{1}{2}\mathbf{u}_t^T \mathbf{R}\mathbf{u}_t + q_t + \partial_{\mathbf{x}} V(\mathbf{x}_t,t)^T \mathbf{f}(\mathbf{x},t) + \partial_{\mathbf{x}} V(\mathbf{x}_t,t)^T \mathbf{G}(\mathbf{x})\mathbf{u}(t) + \frac{1}{2} Tr\left\{\mathbf{G}(\mathbf{x})\Sigma\, \mathbf{G}(\mathbf{x})^T \partial_{\mathbf{x}}^2 V(\mathbf{x}_t,t)\right\}\right] = 0$$

$$\mathbf{u}_t^T \mathbf{R} + \partial_{\mathbf{x}} V(\mathbf{x}_t,t)^T \mathbf{G}(\mathbf{x}_t) = 0$$

$$\boxed{\mathbf{u}_t = -\mathbf{R}^{-1}\mathbf{G}(\mathbf{x}_t)^T \partial_{\mathbf{x}} V(\mathbf{x}_t,t)}$$

Reinforcement Learning Based on Path Integrals

- Sketch of the Path-Integral Derivation

$$-\partial_t V(\mathbf{x}_t,t) = \min_{\mathbf{u}_{t:t_m}} \left[r_t + \partial_{\mathbf{x}} V(\mathbf{x}_t,t)^T \mathbf{F}(\mathbf{x},\mathbf{u},t) + \frac{1}{2} Tr\left\{ \Omega(\mathbf{x},\mathbf{u},t)\, \partial_{\mathbf{x}}^2 V(\mathbf{x}_t,t)\right\}\right]$$

$$\mathbf{u}_t = -\mathbf{R}^{-1}\mathbf{G}(\mathbf{x}_t)^T \partial_{\mathbf{x}} V(\mathbf{x}_t,t)$$

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},t)+\mathbf{G}(\mathbf{x})\big(\mathbf{u}(t)+\boldsymbol{\varepsilon}(t)\big)$$

$$\Downarrow$$

$$\boxed{-\partial_t V(\mathbf{x}_t,t) = -\frac{1}{2}\partial_{\mathbf{x}} V(\mathbf{x}_t,t)^T \mathbf{G}(\mathbf{x})\mathbf{R}^{-1}\mathbf{G}(\mathbf{x})^T \partial_{\mathbf{x}} V(\mathbf{x}_t,t) + q_t + \partial_{\mathbf{x}} V(\mathbf{x}_t,t)^T \mathbf{f}(\mathbf{x},t) + \frac{1}{2} Tr\left\{\mathbf{G}(\mathbf{x})\Sigma\, \mathbf{G}(\mathbf{x})^T \partial_{\mathbf{x}}^2 V(\mathbf{x}_t,t)\right\}}$$

- Sketch of the Path-Integral Derivation

$$-\partial_t V(\mathbf{x}_t,t) = -\frac{1}{2}\partial_{\mathbf{x}} V(\mathbf{x}_t,t)^T \mathbf{G}(\mathbf{x})\mathbf{R}^{-1}\mathbf{G}(\mathbf{x})^T \partial_{\mathbf{x}} V(\mathbf{x}_t,t) + q_t + \partial_{\mathbf{x}} V(\mathbf{x}_t,t)^T \mathbf{f}(\mathbf{x},t) + \frac{1}{2} Tr\left\{\mathbf{G}(\mathbf{x})\Sigma\,\mathbf{G}(\mathbf{x})^T \partial_{\mathbf{x}}^2 V(\mathbf{x}_t,t)\right\}$$

Simplification:

$$\lambda\mathbf{R}^{-1} = \Sigma$$

Log-Transformation Trick:

$$V(\mathbf{x}_t,t) = -\lambda \log \psi(\mathbf{x}_t,t)$$

$$\boxed{\partial_t \psi(\mathbf{x}_t,t) = \frac{1}{\lambda}\psi(\mathbf{x}_t,t)q_t - \partial_{\mathbf{x}}\psi(\mathbf{x}_t,t)^T \mathbf{f}(\mathbf{x},t) - \frac{1}{2} Tr\left\{\mathbf{G}(\mathbf{x})\Sigma\,\mathbf{G}(\mathbf{x})^T \partial_{\mathbf{x}}^2 \psi(\mathbf{x}_t,t)\right\}}$$

Chapman Kolmogorov PDE: 2nd Order and Linear

Reinforcement Learning Based on Path Integrals

- Sketch of the Path-Integral Derivation

$$\partial_t \psi(\mathbf{x}_t, t) = \frac{1}{\lambda}\psi(\mathbf{x}_t, t) q_t - \partial_{\mathbf{x}}\psi(\mathbf{x}_t, t)^T \mathbf{f}(\mathbf{x}, t) - \frac{1}{2} Tr\left\{\mathbf{G}(\mathbf{x}) \Sigma\, \mathbf{G}(\mathbf{x})^T \partial_{\mathbf{x}}^2 \psi(\mathbf{x}_t, t)\right\}$$

Application of Feynman-Kac Theorem:

A numerical method to solve certain PDEs

$$\boxed{\psi(\mathbf{x}_t, t) = E_\tau\left\{\psi(\mathbf{x}_T, T)\exp\left(-\int_{t'=t}^{t'=T} \frac{1}{\lambda} q_{t'}\, dt'\right)\right\}}$$

Reinforcement Learning Based on Path Integrals

- Sketch of the Path-Integral Derivation

$$\psi(\mathbf{x}_t,t) = E_\tau\left\{\psi(\mathbf{x}_T,T)\exp\left(-\int_{t'=t}^{t'=T}\frac{1}{\lambda}q_{t'}\,dt'\right)\right\}$$

$$\mathbf{u}_t = -\mathbf{R}^{-1}\mathbf{G}(\mathbf{x}_t)^T\,\partial_{\mathbf{x}}V(\mathbf{x}_t,t)$$

A bit of algebra ...

$$\boxed{\mathbf{u}_t = E_\tau\left\{w_\tau\mathbf{R}^{-1}\mathbf{G}(\mathbf{x}_t)^T\left(\mathbf{G}(\mathbf{x}_t)\mathbf{R}^{-1}\mathbf{G}(\mathbf{x}_t)^T\right)^{-1}\mathbf{G}(\mathbf{x}_t)\boldsymbol{\varepsilon}_t\right\}}$$

Optimal Control Law

Path Integral RL Applied to Parameterized Policies (Motor Primitives)

- Note that a version of motor primitives can be written as control affine stochastic differential equations

$$\dot{\mathbf{x}} = f(\mathbf{x}) + \mathbf{g}^T(\theta + \varepsilon)$$

  - ε is interpreted as intentionally injected exploration noise
  - the parameters θ are the control vector
  - f(x) is the spring-damper of the primitives
  - g(x) are the basis functions of the function approximator

- It is also necessary to create a iterative version of path integral optimal control
  - the original path integral optimal control framework explores only based on the passive dynamics, i.e., u=0

PI^2 Reinforcement Learning

- For parameterized policies like dynamic motor primitives, a beautifully simple algorithm results:

1) Create K trajectories of the motor primitive for a given task with noise.

2) We can write the cost to go from every time step t of the trajectory as:

$$R_t = q_T + \sum_{i=t}^{T} r_i$$

3) The probability of a trajectory becomes

$$P\left(\xi_t^k\right) = \frac{\exp\left(-\frac{1}{\lambda} R_t^k\right)}{\sum_{j=1}^{K} \exp\left(-\frac{1}{\lambda} R_t^j\right)}$$

4) Update the parameter θ of the motor primitive as

$$\Delta\theta_t = \sum_{k=1}^{K} P\left(\xi_t^k\right) \frac{\mathbf{R}^{-1}\mathbf{g}^k(\mathbf{x}_t)\mathbf{g}^k(\mathbf{x}_t)^T}{\mathbf{g}^k(\mathbf{x}_t)^T \mathbf{R}^{-1}\mathbf{g}^k(\mathbf{x}_t)} \varepsilon_t^k$$

5) Final parameter update

$$\theta^{new} = \theta^{old} + \overline{\Delta\theta_t}$$

Note that there a NO open tuning parameters except for the exploration noise

PI^2 Reinforcement Learning

- The Intuition of Path Integral Reinforcement Learning
  - Generate multiple trials *i* with some variation, e.g., due to noise or exploration
  - For every time t, compute the cost R_t^i for every trial:

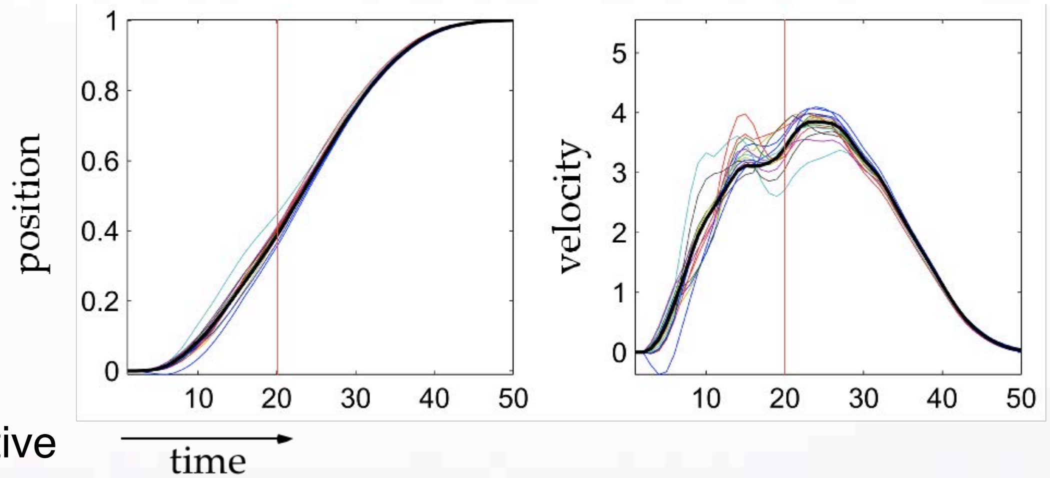
$$R_t^i = q_T + \int_t^T q(\mathbf{x}_t) + \frac{1}{2}\mathbf{u}_t^T \mathbf{R} \mathbf{u}_t d\tau_t$$

  - Convert the cost into a positive weight

$$w_t^i = \exp\left(-\lambda R_t^i\right)$$

  - Update the motor command at every time step to be the reward weighted average of all experienced commands in the trial

$$\mathbf{u}_t^{new} = \frac{\sum_i w_t^i \mathbf{u}_t^i}{\sum_i w_t^i}$$



Surprisingly, this intuition turns out to be the optimal solution

PI^2 Reinforcement Learning: Some Remarks

- PI^2 can be model-based to model-free

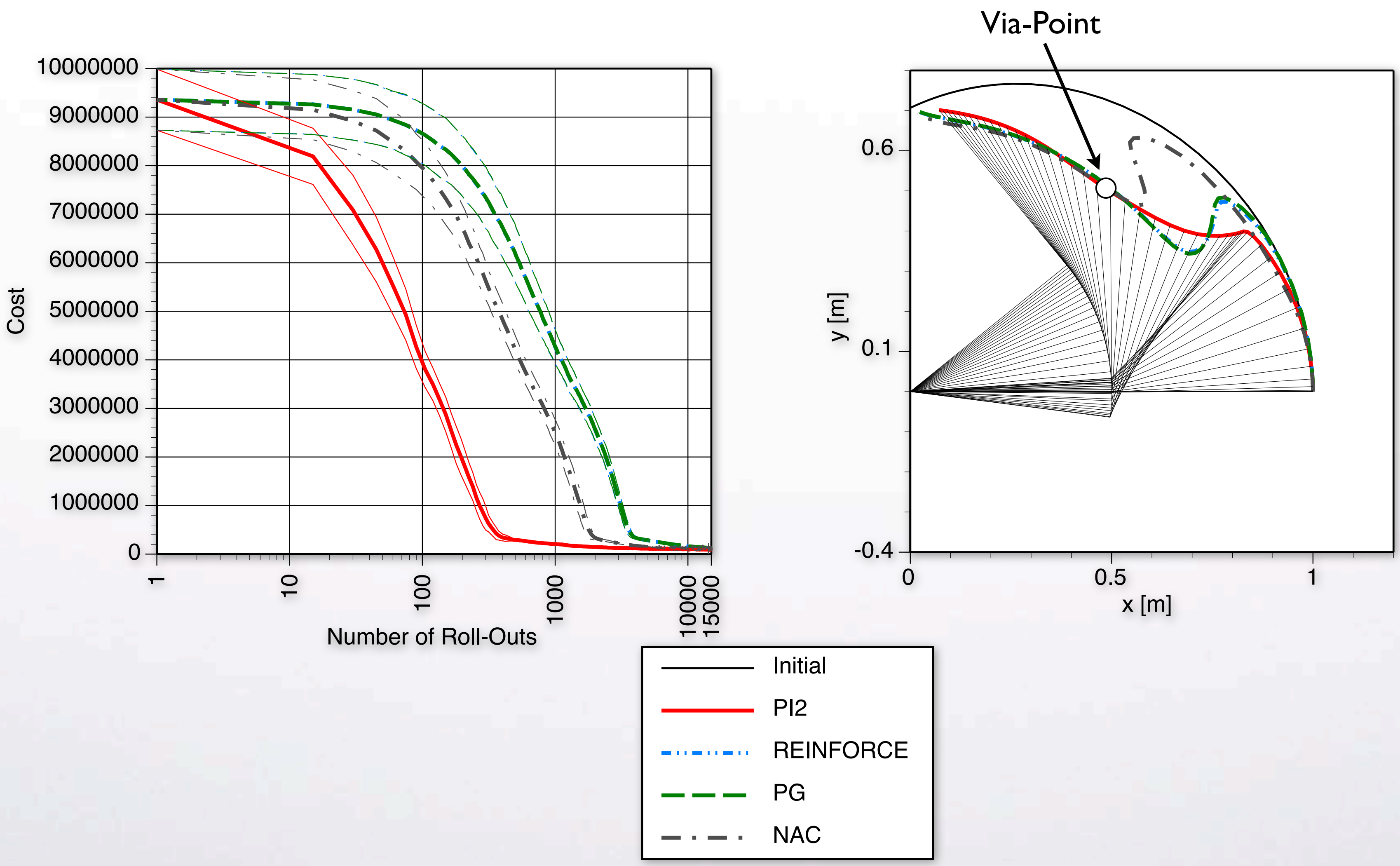
$$\text{Rigid Body Dynamics: } \ddot{\mathbf{q}} = \mathbf{M}(\mathbf{q})^{-1}\left(\mathbf{u} - \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q})\right)$$

$$\text{Control Law: } \mathbf{u} = \mathbf{u}_{ff} + \mathbf{K}_p\left(\mathbf{q}_d - \mathbf{q}\right) + \mathbf{K}_D\left(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}\right)$$

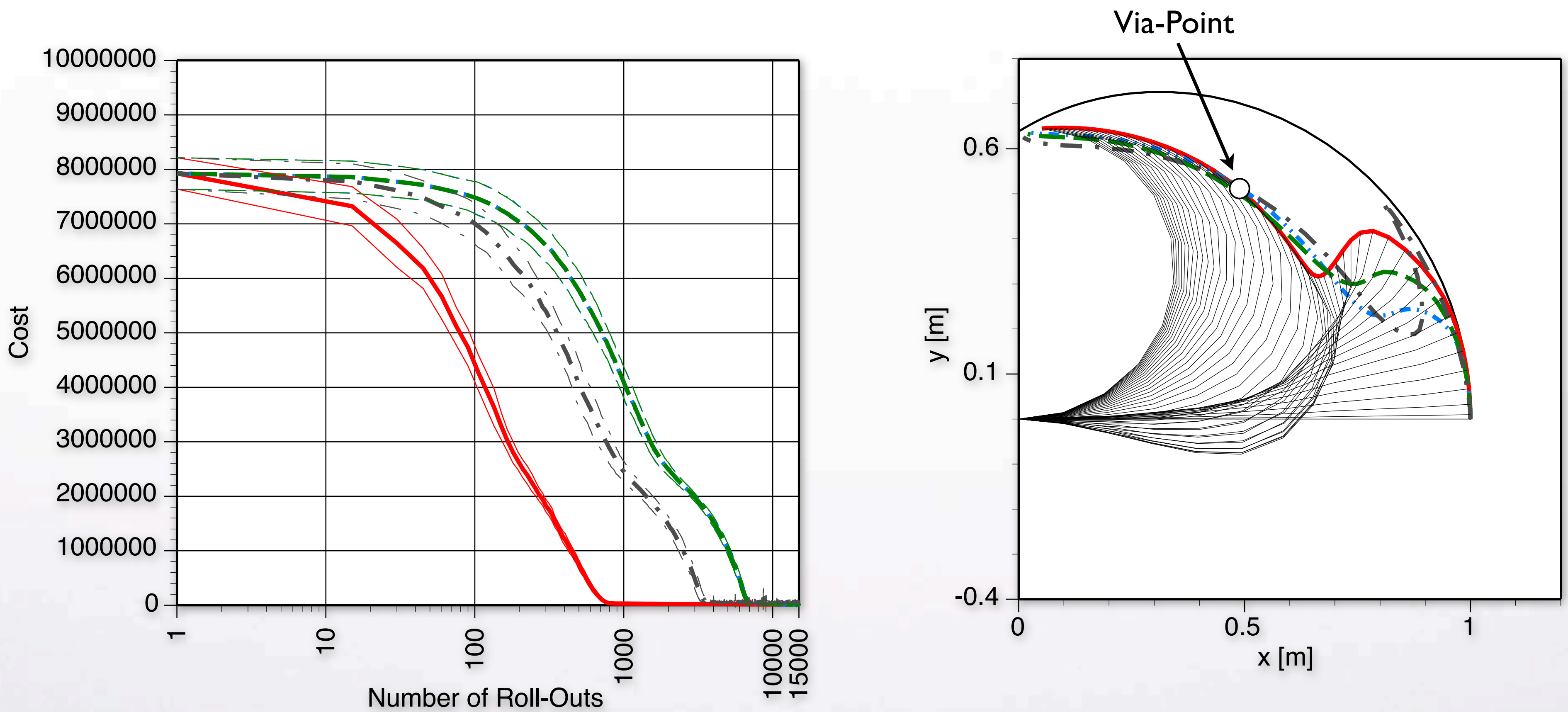
$$\text{Motor Primitives: } \ddot{q}_d^i = \alpha_z\left(\beta_z\left(g^i - q_d^i\right) - \dot{q}_d^i\right) + \boldsymbol{\psi}^T\boldsymbol{\theta}$$

- PI^2 can optimize trajectory plans, controllers, or both
- PI^2 has only one open parameter, i.e., the level of exploration noise
- PI^2 allows a rather simple derivation of inverse reinforcement learning

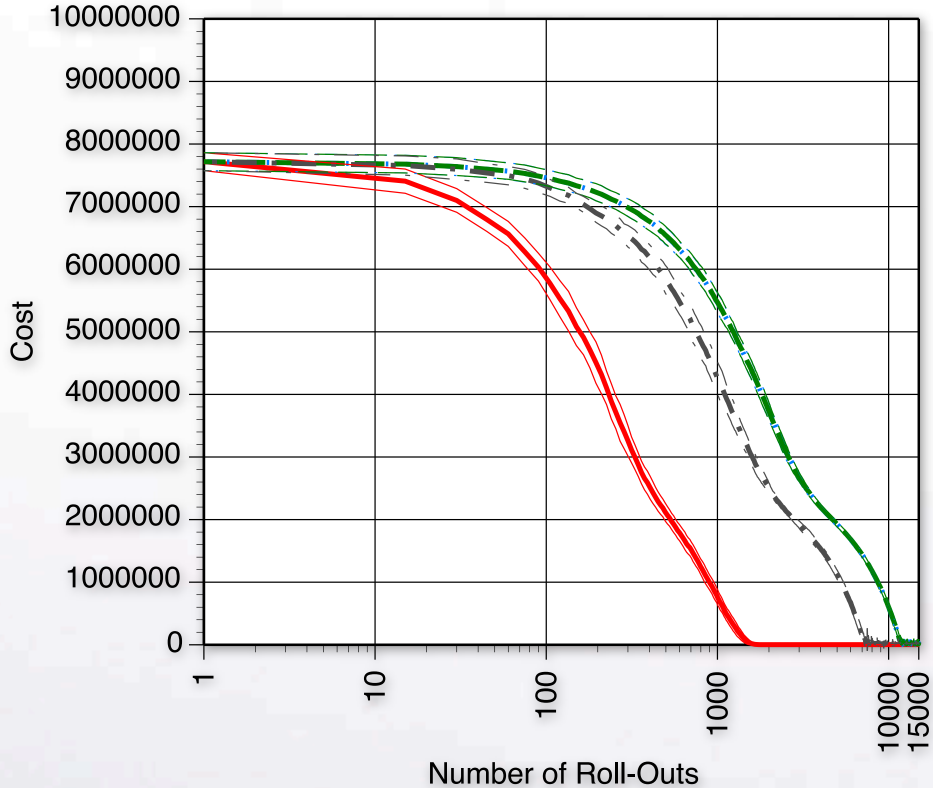
Example: Results on 2D Reaching Through a Via Point

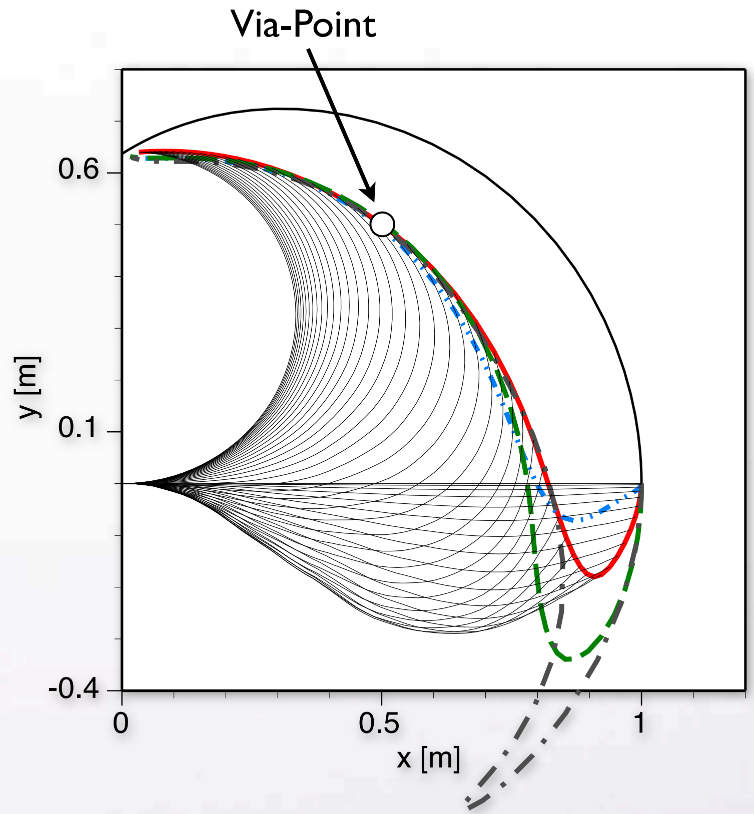


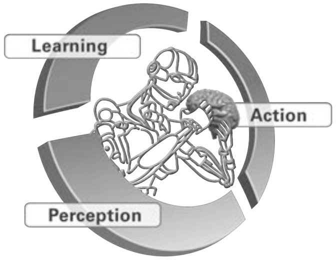
Example: Results on 20D Reaching Through a Via Point



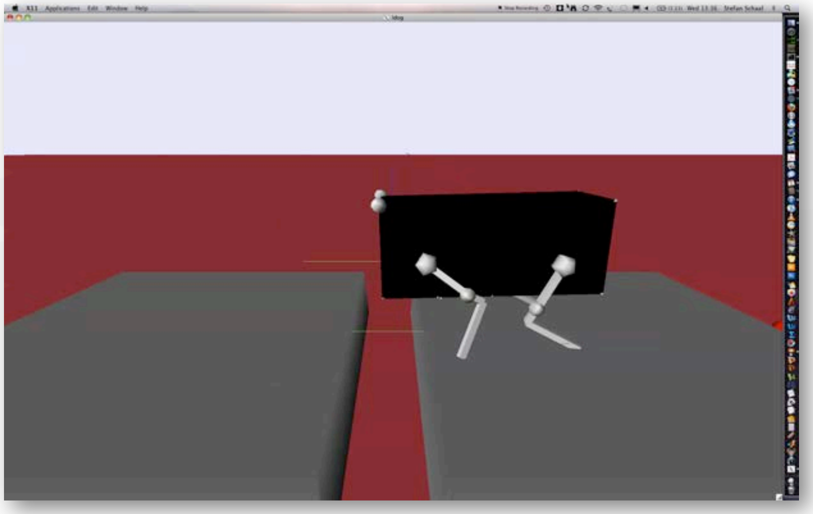
Example: Results on 50D Reaching Through a Via Point

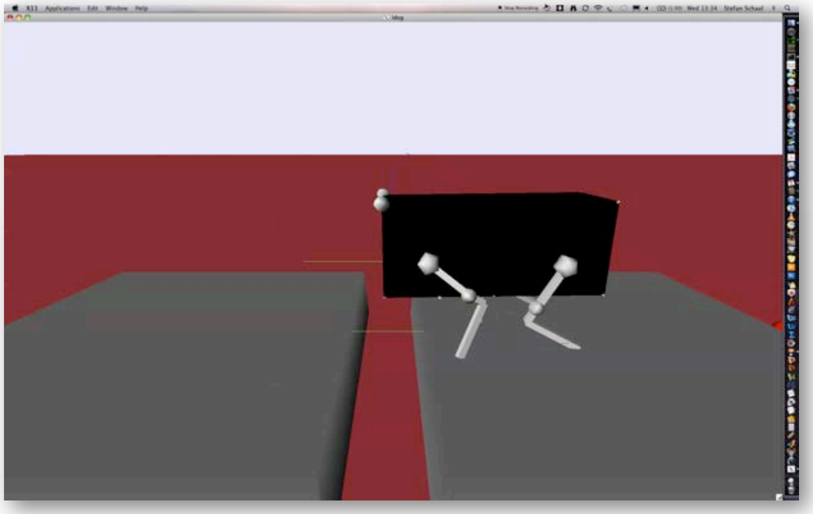




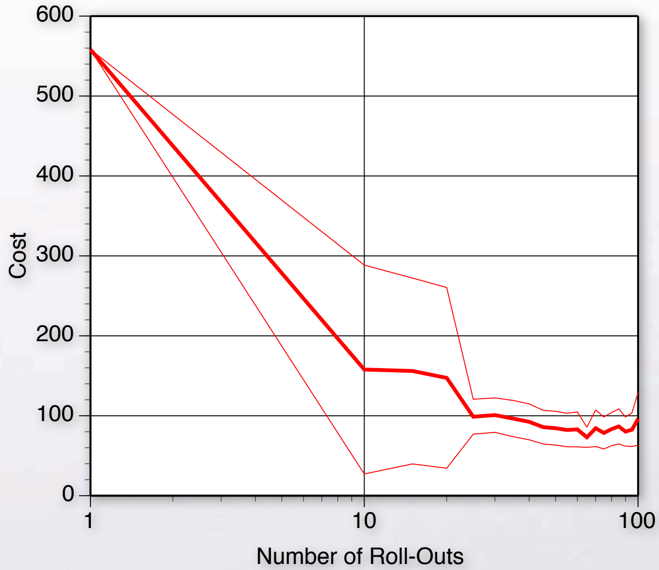


Example: Dog Jump





This is a 12 DOF motor system, using 50 basis functions per primitive. Learning converges after about 20-30 trial! Performance improved by 15cm (0.5 body lengths)



Reinforcement Learning in Manipulation

Peter Pastor   Mrinal Kalakrishnan   Sachin Chitta
Research conducted at Willow Garage

Learning Locomotion over Rough Terrain

Learning Locomotion with LittleDog

http://www-clmc.usc.edu

Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal

Outline

- A Bit of Robotics History
- Foundations of Control
- Adaptive Control
- Learning Control
    - Model-based Robot Learning
    - Reinforcement Learning

What Comes Next?

Towards Truly Autonomous Robots

Very Big Robots

Very Little Robots