Locally Weighted Learning for Control

Alexander Skoglund

Machine Learning Course

AASS, June 2005

Outline

- Locally Weighted Learning, Christopher G. Atkeson et. al. in *Artificial Intelligence Review*, 11:11-73,1997
- Locally Weighted Learning for Control, Christopher G. Atkeson et. al. in *Artificial Intelligence Review*, 11:75-113,1997
  - Temporally independent task
  - Nonlinear optimal control: A simulated puck (similar to the mountain car problem)
- Scalable Techniques from Nonparametric Statistics for Real Time Robot Learning, Stefan Schaal et. al. *Applied Intelligence* 17:49-60, 2002
  - If you want to implement it

What is Local Learning?

What is Local Learning?

“ Local methods assign a weight to each training observation that regulates its influence on the training process. This weight depends upon the location of the training point in the input variable space relative to that of the point to be predicted. Training observations closer to the prediction point generally receive higher weights.” -Jerome H. Friedman

Introduction

- Lazy learning, store data, process when query is required
  - Also called memory based learning
- Relevance of data point is measured as distance
- Locally Weighted Learning (LWL) is suitable for real-time online robot learning because:
  - Fast incremental learning
  - Avoid interference between old and new data (unlearning)

Alternative Techniques

- Nonparametric regression,
- multilayer sigmoidal neural networks,
- radial basis function networks,
- regression trees,
- projection pursuit regression and
- global regression techniques.

An example: Distance Weighting Averaging

- A prediction is based on an average of *n* training examples
- Given a data point (**x**,*y*), relevance $d(\mathbf{x}_i,\mathbf{q})$ is called a distance function, typically the Euclidean distance
- Weighting function also called Kernel function
- Two approaches for weighting:
  - Distance weight directly or
  - Weighting the error criterion

Weighting approaches

- Distance weight directly

$$\hat{y}(q)=\frac{\sum y_i K(d(x_i,q))}{\sum K(d(x_i,q))}$$

- Weighting the error criterion

$$C(\mathbf{q})=\sum_{i=1}^{n}\left[(\hat{y}-y_i)^2 K(d(x_i,\mathbf{q}))\right]$$

when (**q**)=0, C(**q**) is minimized.

Locally Weighted Regression

- Consider a global model trained to minimize:

$$C = \sum_i L(f(\boldsymbol{x}_i, \beta), y_i)$$

- For a global model we usually want to minimize the squared error. If that fails we can:
  - use a more complex global model, or
  - a simple local model of the areas of interest

- We are interested in the latter...

A Linear Global Model

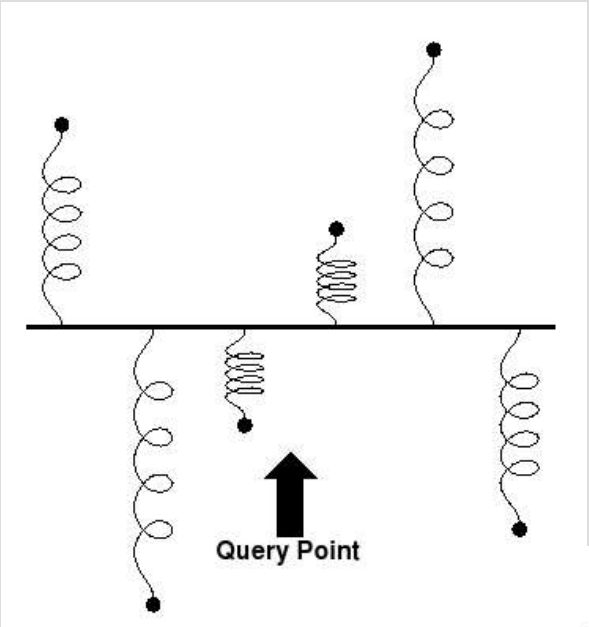
- A global model that has a linear β vector can be written as (each row is one data point, the number of columns is the dimensionality):

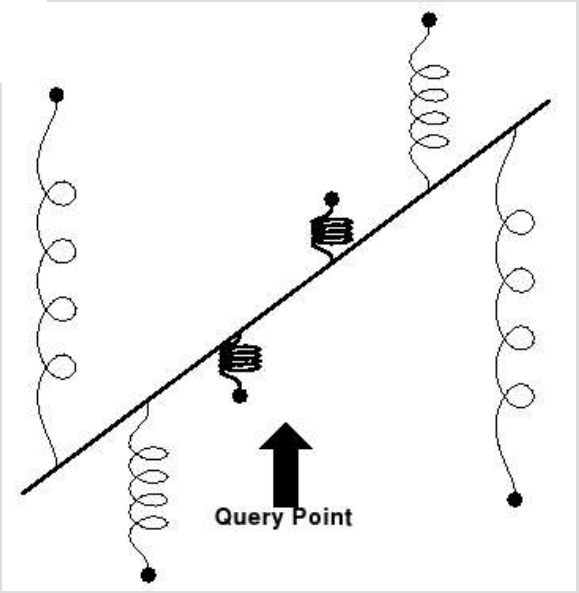
$$X\beta = y$$

- β can be solved for by:

$$(X^T X)\beta = X^T y$$

The Weights as a Physical Interpretation





- In the left figure we see equally strong weights on nearby and far points
- In the right case close points have stronger springs than the far points

Direct Data Weighting

- For simplicity we make the query point q the center point by: $x'_i = [(x_i - q_i)^T \ 1]^T$
- For each data point a distance is calculated from the query point with the weight $w_i = \sqrt{K(d(\boldsymbol{x_i}, \boldsymbol{q}))}$
  - Written in matrix form: $z_i = w_i x_i$ => $\mathbf{Z = WX}$
  - Written in matrix form: $v_i = w_i y_i$ => $\mathbf{v = Wy}$
- Solving for b using these new variables:

$$(\boldsymbol{Z}^T \boldsymbol{Z})\beta = \boldsymbol{Z}^T \boldsymbol{v}$$

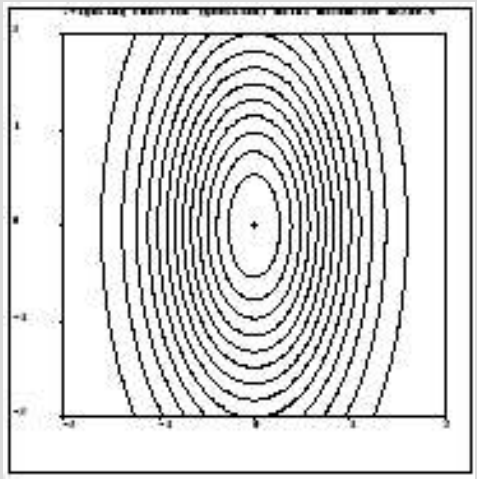
Distance Functions

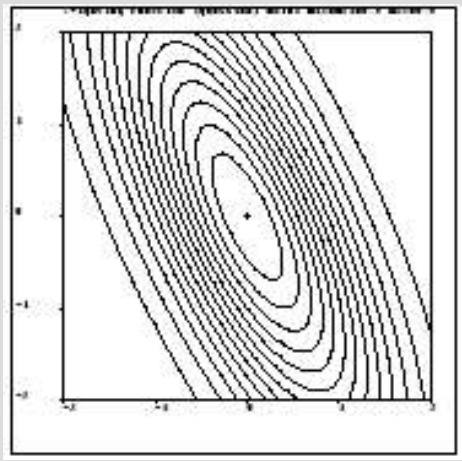
- Where to use a distance function:
  - Same on all data: Global *d()*
  - Set by the query point: Query-based local *d()*
  - Different *d()* measure for different points (chosen in advance of the query, not dependent of the query)

- The importance of a given input dimension is *scaled* by the distance function:
  - If all scaling factors are non-zero this might lead to better predictions
  - If some dimensions are set to zero, the model is *global* in that dimension

Distance Functions for Continuous Inputs

- Different distance functions:
  - Unweighted Euclidean Distance
  - Diagonally weighted Euclidean Distance
  - Fully weighted Euclidean Distance
  - Unweighted Lp norm (Minkowski metric):
  - Diagonally weighted and fully weighted Lp norm
- Is represented by the matrix **M**
- For symbolic inputs see Atkeson 1996



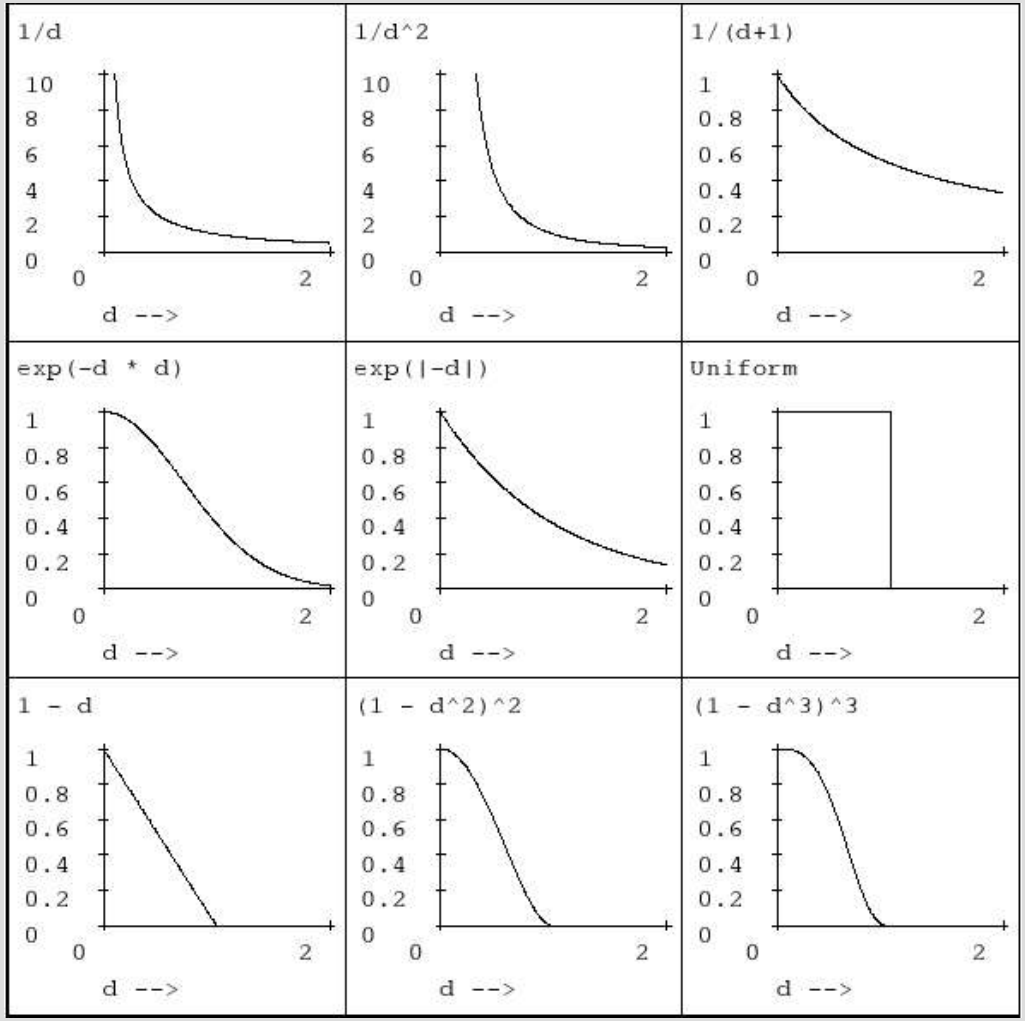


Smoothing Parameters

- The smoothing parameter defines the scale over where generalization is performed. The different selection types are:
  - Fixed bandwidth (large variations, undefined if no data)
  - Nearest neighbor bandwidth
  - Global bandwidth
  - Query-based local bandwidth
  - Point-based local bandwidth, preferred over Query-based because it allows rapid asymmetric changes in data
- Represented by parameter *h*

Weighting Functions

- Different shapes of the kernel weighting function
- Look up for regions with no data!
- There is no evidence that the selection is critical
- K(d) represents the weighting



Ridge Regression

- If a query point is in a region with a small number of data points nearby, the β parameter can not be solved for.

- Then use “Ridge Regression”, which can be said to be adding fake data to regions with no or insufficient data.

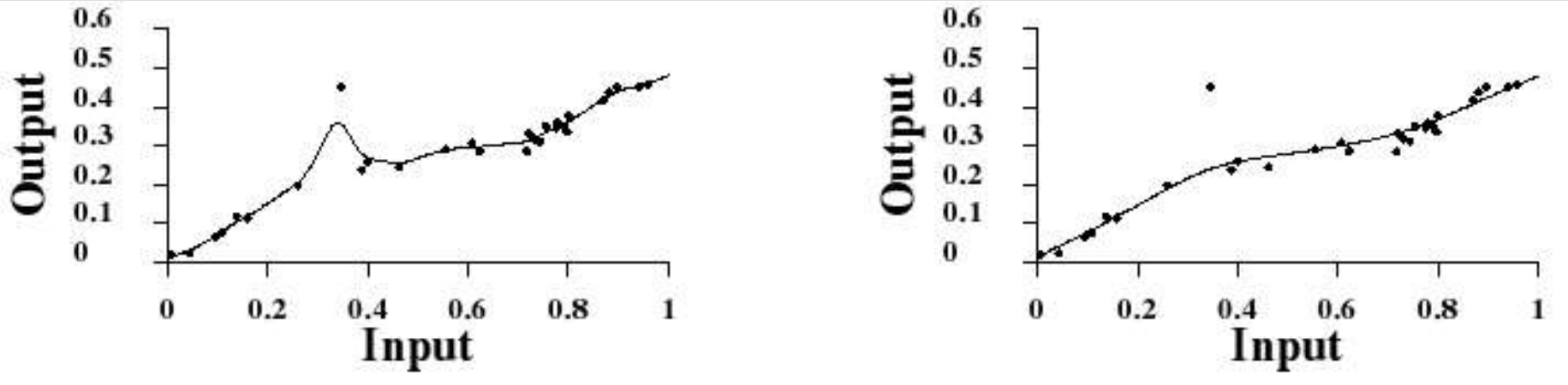
Dimensionality Reduction

- To identify dimensions with no or little data one can use:
  - Locally weighted PCA to identify manifolds of data (SVD of $\mathbf{Z}$ matrix)
  - SVD and set small singular values (of the inverse of $\mathbf{Z}^T\mathbf{Z}$) to zero, this means we eliminate those directions

Predictions

- Possible to estimate prediction error
  - Variance is estimated as the sum of squared errors divided by the sum of squared (modified) weights
  - Bias = E((q))-y_{true}(q)
- Leave-one-out cross validation is easy to perform in memory based methods (in contrast to methods with a training stage.)
  - Pretend that data point x_i is not seen and calculate how well the remaining data points can predict y_i

Identify Outliers



- Attach a weight to each data point
  - base on cross validation of all data (global case)
  - at query time, generate the weight by cross validation of nearby points
  - Use Robust Regression by a weight called

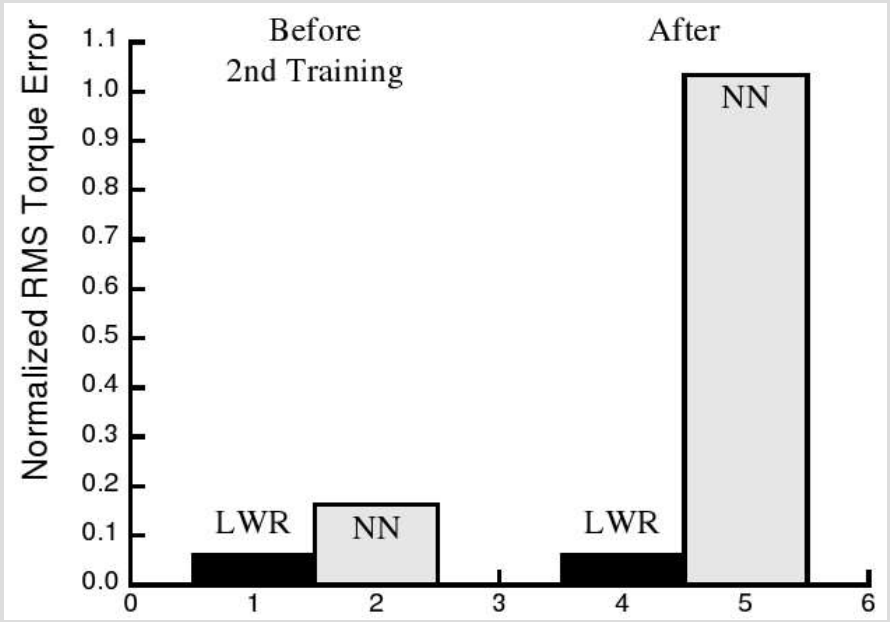
    $u_i \Rightarrow w_i = u_i K(d(\boldsymbol{x}_i, \boldsymbol{q}))$

Tuning

- Parameters to tune:
  - Bandwidth or Smoothing parameter, h
  - Distance metric, d()
  - Weighting or Kernel function, K()
- In addition we have:
  - Ridge regression
  - Outliers thresholds
- Ways to tune:
  - Global (minimize cv errors)
  - Query-based local
  - Point-based local (different for each point)
- How to:
  - Plug-in approach
  - Optimization

Interference

- The most important motivation for using LWL!
- To illustrate the problem we compare a ANN with LWL on learning the inverse dynamics of a two link arm.



Key issues

- Is LWL a better FA than, let say, RBF or sigmoidal neural network?

  - No, but it *avoids negative interferences* with old data!

- LWL forms the model after the query is given.

Temporally Independent Tasks

- y = f(x,u) + noise
  - u is the action, x the observed state, y outcome
  - f() is not known (we learn a model of f() called f'())
- The inverse model is $u=f^{-1}(x,y)$
  - During training each training sample is stored in a data base, < x(i),y(i)->u(i) >
  - We can then match our query with the data base, nearest neighbor, interpolation etc...
- The forward model y=f'(x,u)
  - can be used to perform a “mental simulation”
  - The data base will be < x(i),u(i)->y(i) >

Forward model

- To choose action we must perform a search:
  - Grid Search
  - Random Search
  - First Order Gradient Search
  - Second Order Gradient Search
- The inverse model can be used to provide the forward model with a starting point for the search.

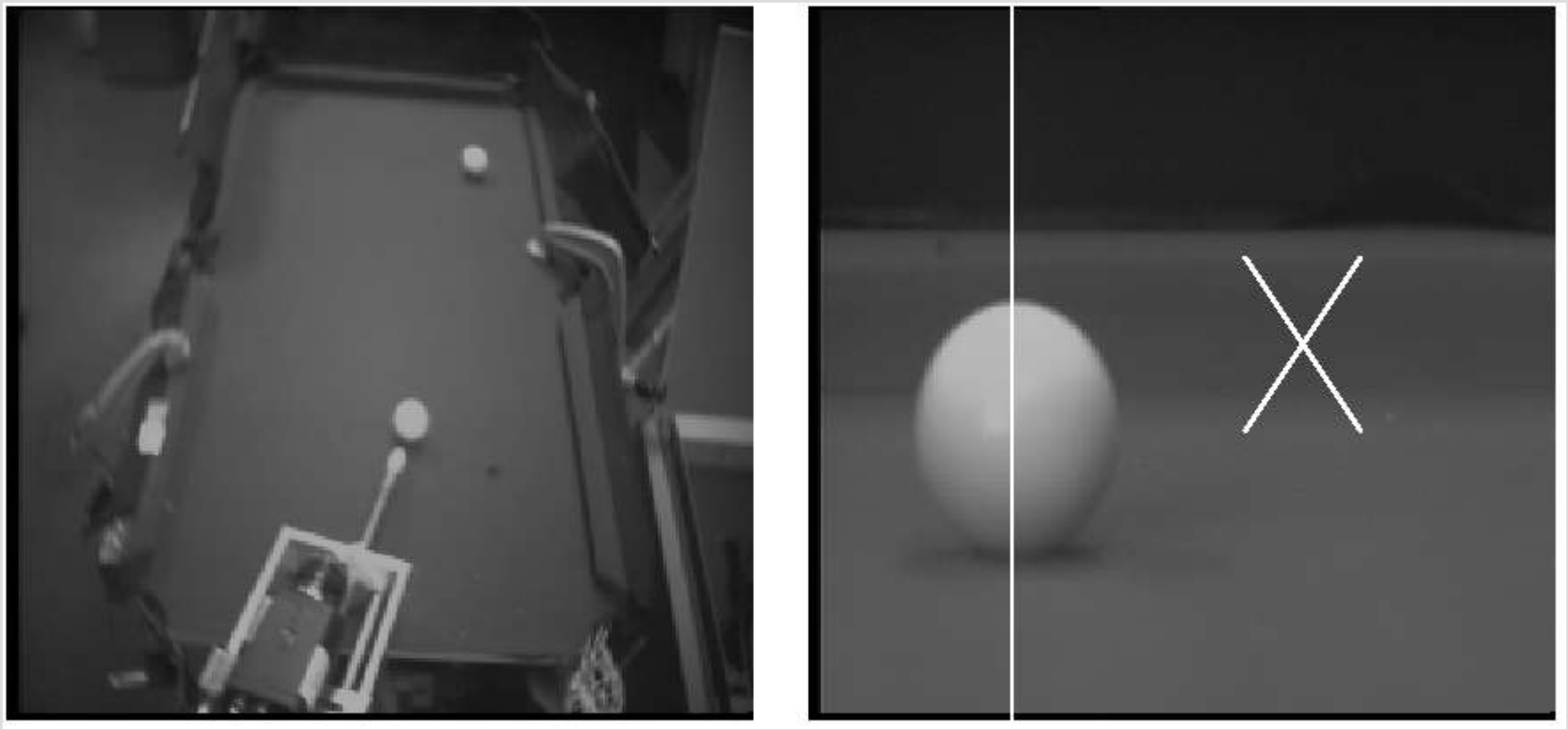
Billiard

- Small pool table, a spring actuated cue
- The only sensors are two cameras: one along the cue, the second has a top view of the pool table.
- Each shot proceed as follows:
  - Each shot start at the same position, i.e., the cue ball at the same position every time
  - The ball we want to sink is placed randomly, this is the state *(x,y)*
  - Use the inverse model followed by a search of the forward model to find the action u, predicted to sink the ball

Billiard, 2

  - The outcome is the cushion and position (on the cushion) where the ball makes the first hit.
  - The memory base (data base) is updated with the new observation
- Both the forward and inverse model uses the locally weighted regression with outlier removal and cross validation for choosing kernel width.
- The forward and inverse models where used together
- To work the action requires 1%(??) accuracy or better, which the LWR provides

Billiard, 3



The cue swivels until the centroid of the object (shown by the vertical line) coincides with the chosen action (the cross).

Performance

- With less than 100 trails, a 75% success is achieved (equally to an MIT billiard champion), some shots are “virtually impossibly difficult”.

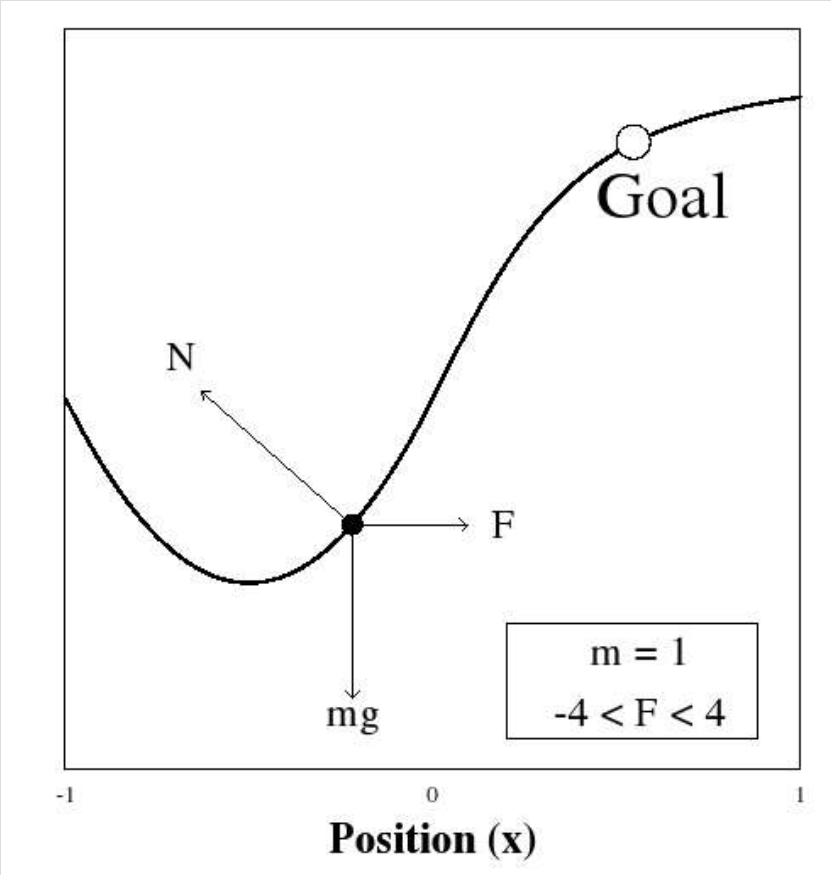
- One reason for success:
  - Non-uniform data distribution, all training data are clustered around the state-actions pairs that succeeded. Little exploration was made.

Nonlinear Optimal Control

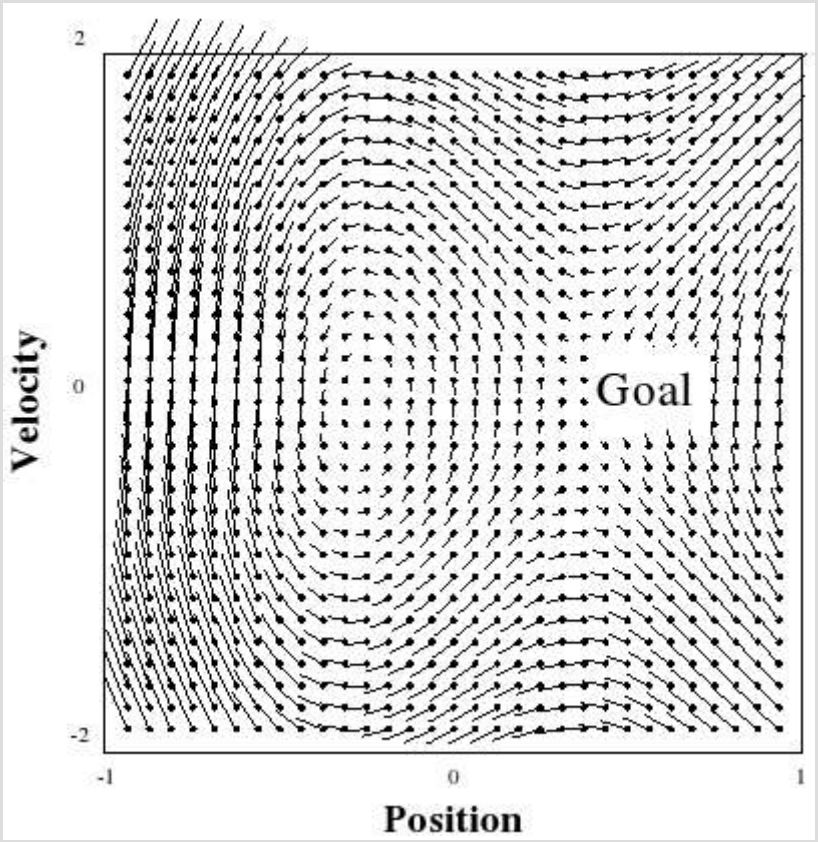
- We need a global model (control law) based on many local models
- Given a cost function:
  $g(t) = G(x(t), u(t), t)$
- The task is to minimize the total cost
- This gives us a delayed reward function, since the action taken at time t affects all the subsequent states. Reinforcement learning!

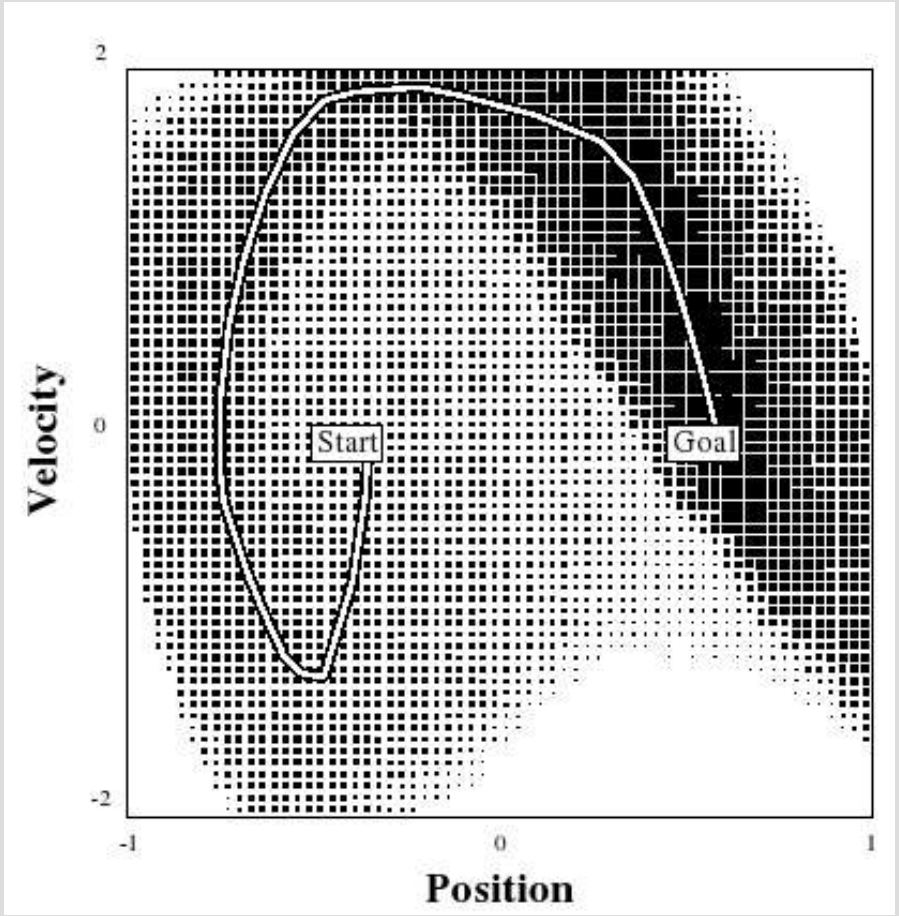
Example 2: The puck

- Objective: drive the puck up the hill to the goal in minimum number of time steps.
- State is the position, x, and velocity, x'
- Constrains: maximum force is $-4 \leq a \leq 4$, a is our action
- No friction!



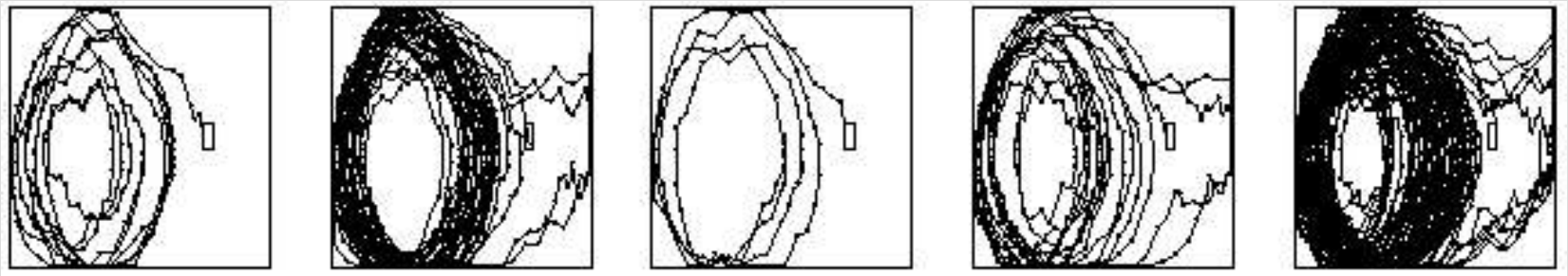
Example 2: The optimal solution





Example 2: Reinforcement Learning

- Discretized state space, a grid 60x60 units for the forward model and the value function.
- Actions, discretized into 5 levels, (-4,-2,0,2,4 N)
- All transitions where remembered
- Exploration is achieved by giving unvisited states the cost of zero (visited bad states will be given negative values)



Example 2: Reinforcement Learning

- Same as before except that:
  transitions between cells were filled in by predictions from LWR forward model. This means that unvisited transitions were extrapolated from actual experience.

- The value function is a table in both cases.

- Much faster than than before in terms of *trails*.