A Framework for RL Learning Controllers

Martin Riedmiller
Arbeitsgruppe Neuroinformatik
Universität Osnabrück

Martin.Riedmiller@uos.de

Motivation

- driven by the idea to have a self-learning control system that completely learns from scratch. Only actions/ sensor inputs/ goals & constraints are specified $\Rightarrow$ Reinforcement Learning, Neuro Dynamic Programming
- no prior policy, no prior model $\Rightarrow$ Q-learning.
- continuos sensor values $\Rightarrow$ Neural network (here: MLP) (reason: continuous, generalisation. other models might exist ;-))
- fast learning: data-efficient directly applicable to real systems: $\Rightarrow$ Neural Fitted Q Iteration (NFQ), (Riedmiller, 2005)
- standard software module: robust, easy to use (e.g. parameters), 'off-the-shelf'-method

Talk outline

- short intro to NDP and NFQ
- current developments
- examples
- open questions

Idea: Sharpen the core RL method by doing a wide range of (real-world) applications

Neuro Dynamic Programming in a Nutshell

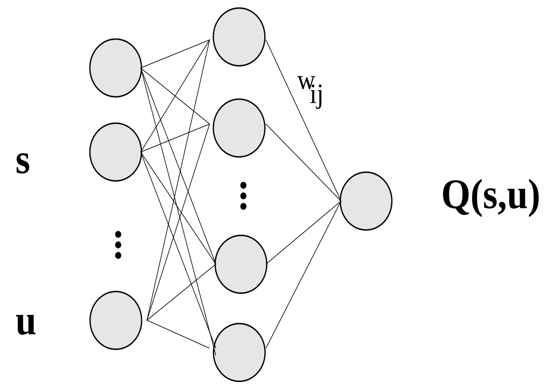
- MDP: states, actions, (probabilistic) transition model, costs for transitions
- learning goal: find a policy (controller) π to minimize the expected summed trajectory costs:

$$J^*(s) = min_\pi E(\sum_t c(s_t, pi(s_t))|s_0 = s) \text{ for all } s$$

- choice of transition costs $c(s, u)$ specifies control goal, very flexible (LQR is a special case);
- Value Iteration: approximate J^* iteratively ('learning'); optimal policy can be derived thereof
- Q-learning: model free learning

Neural Value Functions (MLP)

- multilayer perceptron stores value function $J : X \to \Re$ (or $Q : X, U \to \Re$ respectively)
- works for continuous or very large state spaces
- only few parameters determine function over complete state space: generalisation



but: if used with classical online RL methods, learning is very slow!
Reason: non-local approximation: unlearning the function

Idea NFQ: Explicit memorisation of all observations of 'state - action - successor state' tuples:
$\Rightarrow$ set of memorized system transitions (s, a, s')

Neural Fitted Q Iteration (NFQ) (Riedmiller 2005, Ernst 2005)

$(s_1, a_1, s'_1), \ldots, (s_N, a_N, s'_N)$ are sampled transitions (state-action-successor state).

$Q_k \in \mathcal{F}$: approximation of Q-value function at state k

For each transition sample $1...N$ do

$$\hat{Q}(s_i, a_i) := c_i(s_i, a_i) + \gamma min_a Q_k(s'_i, a)$$

Compute next iterate Q_{k+1} by

$$Q_{k+1} = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} |f(s_i, a_i) - \hat{Q}(s_i, a_i)|^2$$

$\Rightarrow$Q- Value Iteration becomes a series of batch supervised learning problems

Some aspects of using NFQ as a core RL module

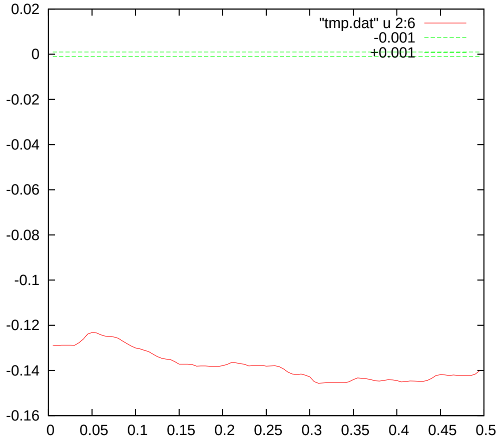
remember goal: efficient, robust, simple to use, no hidden prior knowlege

- Rprop for batch supervised learning: very fast, practically parameter free
  (Riedmiller, 1992)
- standard cost fomulation for set point regulation problems
- forced output values
- sampling the data

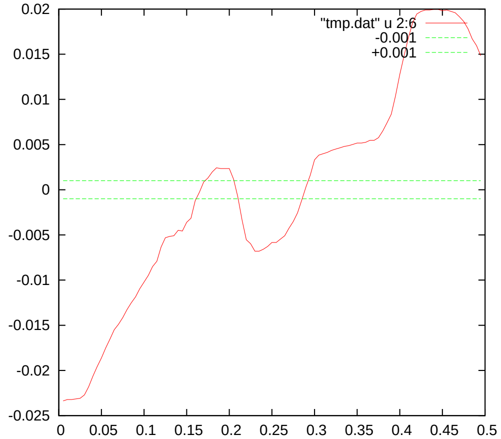
Set Point Regulation

set point regulation: bring one or more sensor values to a desired target value.
notation: $s \in X^+ \Leftrightarrow$ all sensor values are at their target values ($\pm$ tolerance)
Proposed standard cost function (optimizes time to goal region):

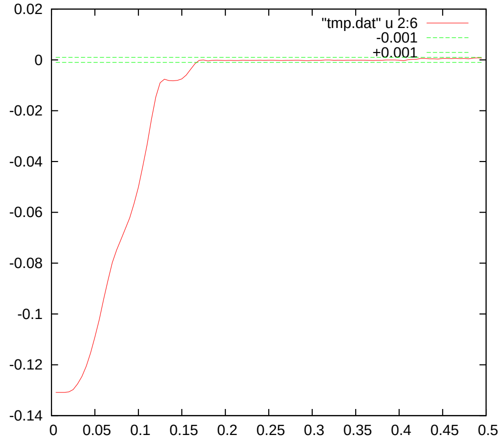
$$c(s,u) = \begin{cases} 0 & , \quad \text{if } s \in X^+ \\ 0.01 & , \quad else \end{cases}$$



bad



improved



good

Set Point Regulation continued

Proposed standard cost function (optimizes time to goal region):

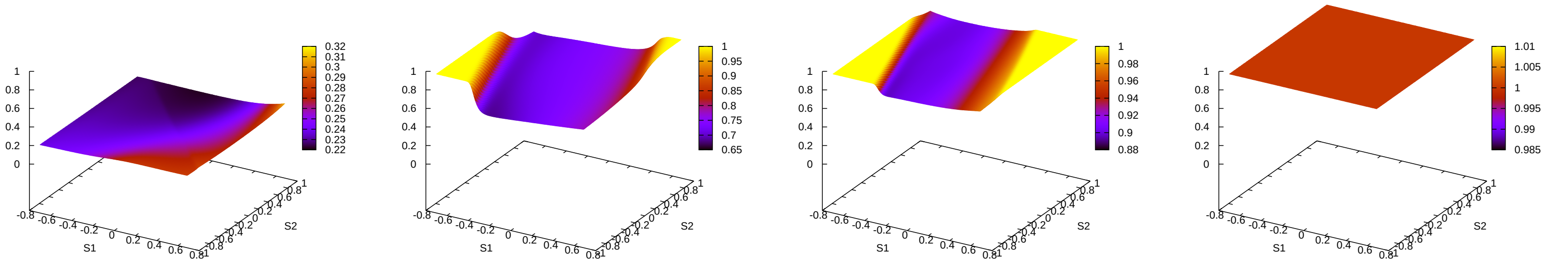
$$c(s,u) = \begin{cases} 0 & , \quad \text{if } s \in X^+ \\ 0.01 & , \quad else \end{cases}$$

- A succesful policy must learn to actively bring and *keep* state to X^+ (no explicitly fixed terminal state)
- good: no prior knowledge: the controller must find out by itself how to control the system so that it not only reaches X^+ but also can be kept there!
- problem: danger of MLP output continuosly increasing due to generalisation effects (since with the above costs rule, for all the state-action pairs, the Q-value is at least as high as its successor state).
- idea: for some states within terminal area we know that $J^*(s) \stackrel{!}{=} 0$. Idea: Explicitly force NN output to target values 0 at some of those states.

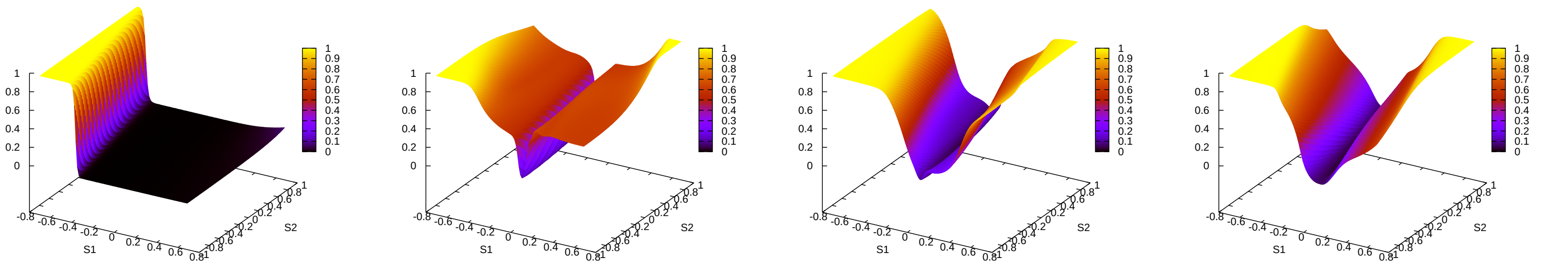
Forced Output Values

Idea: introduce artificial training patterns with forced target value $J^*(s) = 0$.
candidates: use states in the center of target area. Alternatively: do tests.

Example Cart-Pole Balancing. No Forced Output Values:



With forcing:



Sampling transitions

Learning the Q-funtion by NFQ happens offline. The online information required from the controlled system are transition triples state-action-successor state. These are valid indpendend of the used policy $\Rightarrow$ High flexiblity in transition sampling.

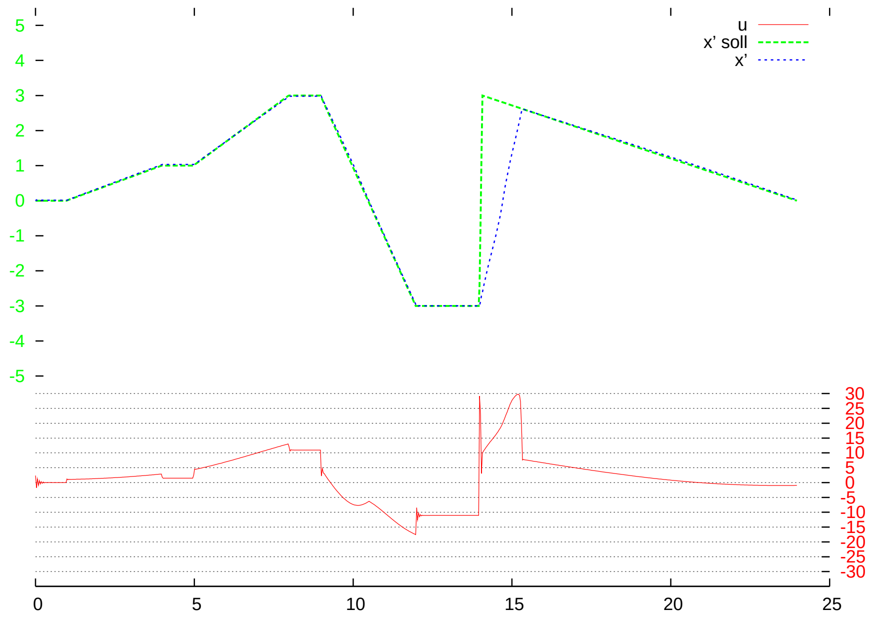
Sampling transitions

Learning the Q-funtion by NFQ happens offline. The online information required from the controlled system are transition triples state-action-successor state. These are valid indpendend of the used policy $\Rightarrow$ High flexiblity in transition sampling.
Methods:

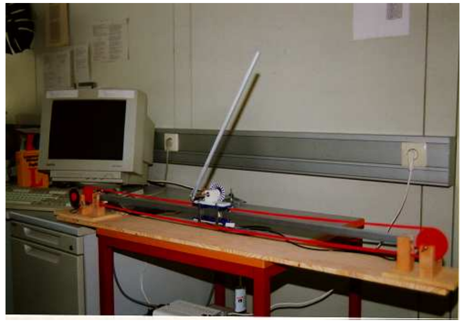
- interleaved mode (sample, learn, sample, learn,...)
- sample then learn (e.g. if sampling requires human interaction)
- using prior knowledge, e.g. an existing controller to sample
- 'task shaping': learn a different (maybe simpler) task first. Example: cart-pole balancing/ suspension
- sampling on multi-time scales
- policy screening: keep succesful policy active for some time to test and/or collect particularly interesting transitions

Further aspects of using NFQ as a core RL module

- input representation, e.g. dealing with delays
- learning modules and methods, e.g. advantage updating, residual gradient methods
- policy representation, e.g. continuous outputs (Hafner, 2008)



Real Cart Pole



task: real cart pole, starting downwards, setpoint regulation (position, pole angle)
state dim: 4 (cart and pole position, cart and pole delta position)
actions: 3 (negative voltage, positive voltage, 0)
$\triangle t$: 30 ms, 5 - 10 - 10 - 1 MLP

- standard set-point cost function

$$c(s,u) = \begin{cases} 0 & , \text{ if } |\theta| < 3^o, |pos| < 0.05m \text{ ('setpoint area')} \\ 0.01 & , \; else \end{cases}$$

$$Q(s,a) = 1, \text{if } |pos| > 0.25m \text{ (constraint)}$$

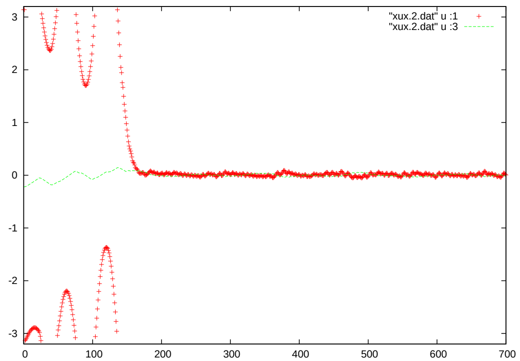
- forced output value '0' at (0,0,0,0)

- interleaved learning sampling

Real Cart Pole - Results

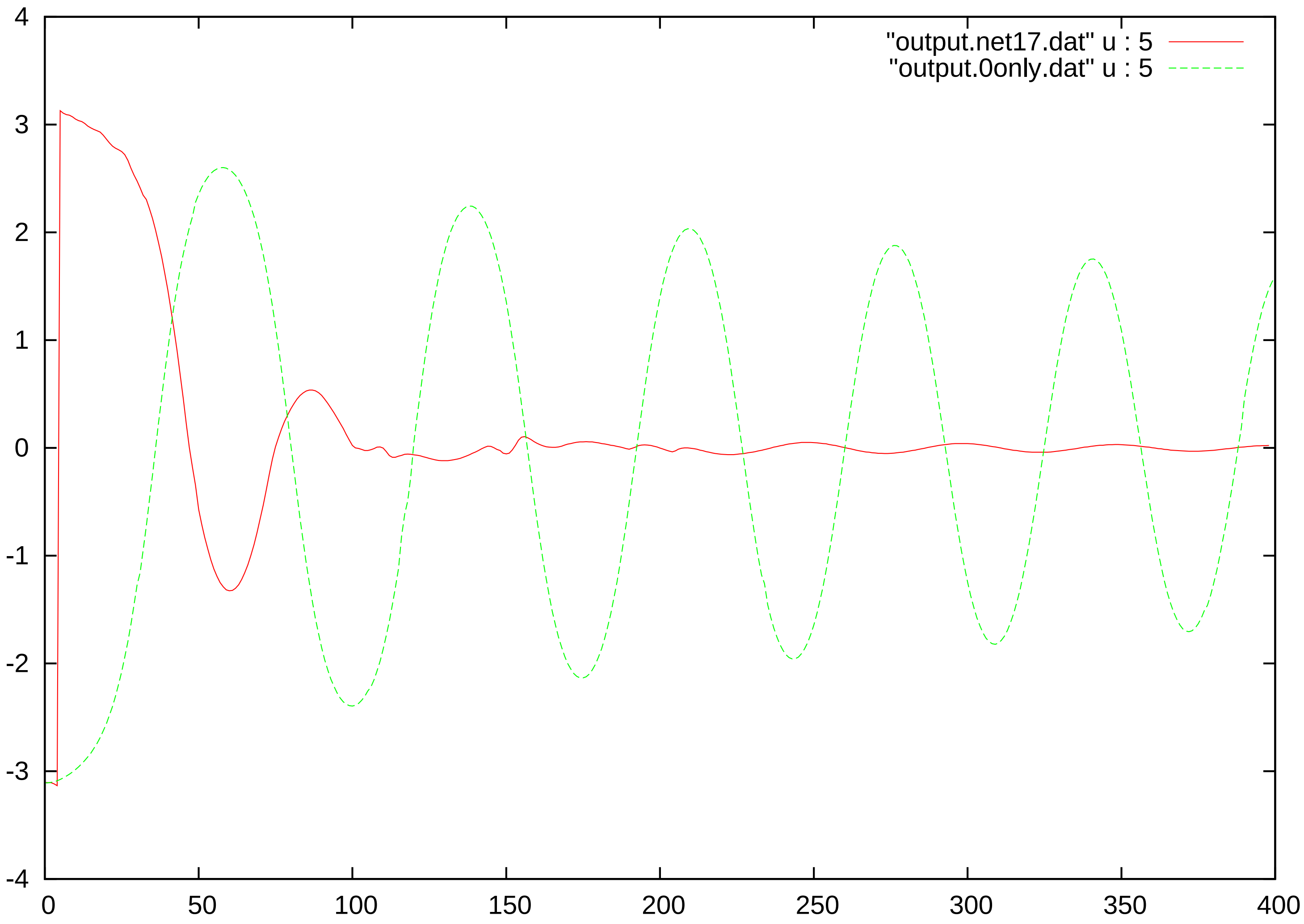
- learns directly with the real system, completely from scratch
- one single NN controls both swingup and balancing
- no human interaction during learning process required
- pure interaction time is much less than 1 hour (300 trials of max. 12s)
- overall learning time is less than 10 hours
- sampled data can be reused to learn completely different task, e.g. suspension

Video



Real Cart Pole Suspension - Results

Reuse of the transition data of the previous trial. < 50 NFQ iterations.



Neuro Dribbling for a soccer robot

Task: 'Dribbling:' Moving to a target direction without loosing the ball.

state dim: 5 (rel. robot speed (x,y), rotation speed, delta angle to target direction, ballposession)
actions: 4 (pairs of target speeds in forward and sideward direction)
$\triangle t$: 33 ms

- offline sampling: random sampling (100 episodes) - NFQ (100 iterations) - greedy sampling (100 episodes) - NFQ (100 iterations)
- standard set-point cost function

$$c(s,u) = \begin{cases} 0 & , \quad \text{if } |\theta - target| < 5^o \ \text{('setpoint area')} \\ 0.01 & , \quad else \end{cases}$$

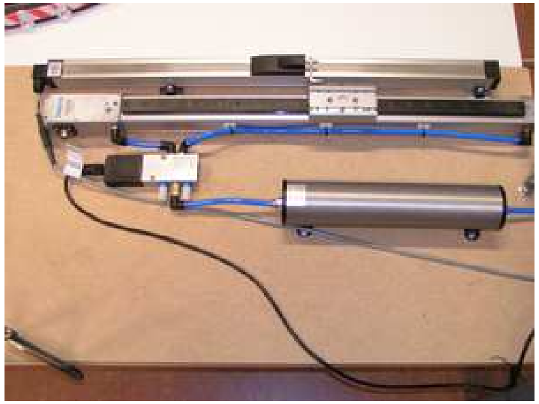
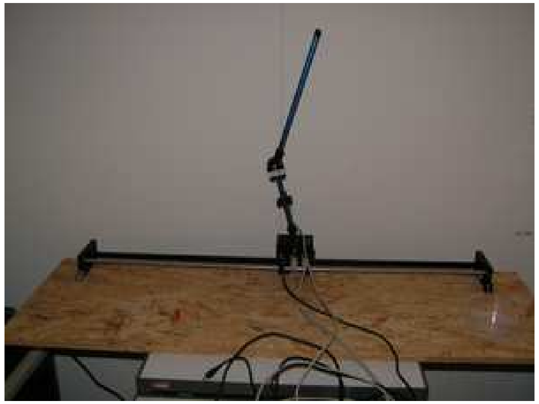
$Q(s,a) = 1,$ if ball is lost (failure)

Neuro Dribbling Results

- decent human interaction in two phases of data sampling of about 15 minutes each.
- two offline NFQ learning phases of about 3 hours each
- used for dribbling in Brainstormer's World Champion Team RoboCup 2007
- won Technical Challenge Award RoboCup 2007

Some current applications

- Real Cart Pole, real Cart Double Pole
- Neuro Dribbling Soccer Robot (2007)
- Neural steering of autonomous car (Riedmiller et.al, Fbit 2007)
- Active Damping of a convertible car (accurate FEM Simulation, Industrial Project 2006-2007)
- Real slot car racing (industrial project for Hannover Fair 2008)
- Pneumatic positioning (since 2008)

Summary

- already considerable robustness w.r.t network structure, learning parameters, ...
- efficiency: can be applied to real systems directly
- standardized framework helps to quick start (e.g. pneumatic positioning project needed only one afternoon to obtain first good controllers)
- still (moderate) experience required to determine time interval, tolerance band, action set, ... $\Rightarrow$ further research

Ongoing and future work

- extending the framework: continuos actions, policy representation
- active learning/ data selection
- incorporating prior knowledge
- cooperative Multi-agent systems: scheduling (Gabel 2008)

Open challenges:

- convergence issues
- proof of stability of learned controllers
- other regression methods, e.g. Gaussian Processes
- combination with other methods, e.g. policy gradient,...