Supervised learning based sequential decision making

Damien Ernst

Department of Electrical Engineering and Computer Science
University of Liège

ICOPI - October 19-21, 2005

.

Find slides: http://montefiore.ulg.ac.be/∼ernst/

Problem addressed in this talk

How can we design algorithms able to extract from experience good sequential decision making policies?

$\Rightarrow$ Discuss different algorithms exploiting batch-mode supervised learning.

$\Rightarrow$ Illustrate one of these algorithms, named fitted Q iteration, on an academic power systems example

---

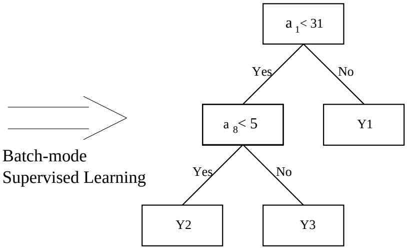
NB.

- Many practical problems are concerned (medical applications, robot control, finance, ...)
- Most are related to complex and uncertain environments
- To simplify derivations, we restrict to the deterministic case

Batch-mode supervised learning

- From sample $ls = (x, y)^N$ of N observations (inputs, outputs)
- Compute a model $\hat{y}(x) = f^{ls}(x)$ (decision tree, MLP, ...)

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	Y
60	19	18	17	0	1	1	1	Y1
60	3	22	23	1	29	11	23	Y1
75	9	2	1	3	77	46	3	Y1
2	10	10	2	234	0	0	0	Y2
3	7	9	18	5	0	0	0	Y2
2	14	5	10	8	10	8	10	Y3
65	3	20	21	2	0	1	1	?



$x = (a_1, \ldots, a_n)$; y real number or class-label

Learning a static decision policy

Given state information x and a set of possible decisions d, learn an approximation of the policy $d^*(x)$ maximizing the reward $r(x, d)$.

To learn, let's assume avaible a sample of N elements of the type (x, d, r).

Case 1: learn from a sample of optimal decisions $(x, d^*)^N$:
$\Rightarrow$ direct application of SL to get $\hat{d}^*(x)$.

Learning a static decision policy

Given state information x and a set of possible decisions d, learn an approximation of the policy $d^*(x)$ maximizing the reward $r(x, d)$.

To learn, let's assume avaible a sample of N elements of the type (x, d, r).

Case 1: learn from a sample of optimal decisions $(x, d^*)^N$:
$\Rightarrow$ direct application of SL to get $\hat{d}^*(x)$.

Case 2: learn from a sample of random decisions $(x, d, r)^N$:
$\Rightarrow$ apply SL to get $\hat{r}(x, d)$ and compute $\hat{d}^*(x) = \arg\max_d \hat{r}(x, d)$.

Learning a sequential decision policy

Given initial state information x_0, state dynamics $x_{t+1} = f(x_t, d_t)$, and instantaneous reward $r(x, d)$, find $(d_0^*, \ldots, d_{h-1}^*)$ maximizing the cumulated discounted reward over h stages

$$R(x_0, d_0, \ldots, d_{h-1}) = \sum_{t=0}^{h-1} \gamma^t r(x_t, d_t).$$

Different kinds of optimal policies:

Open loop: $d_t^* = d^*(x_0, t)$ (OK in the deterministic case)

Closed loop: $d_t^* = d^*(x_t, t)$ (Also OK in the stochastic case)

Stationary: $d_t^* = d^*(x_t)$ (OK in the infinite horizon case)

Learning a sequential decision policy

Given initial state information x_0, state dynamics $x_{t+1} = f(x_t, d_t)$, and instantaneous reward $r(x, d)$, find $(d_0^*, \ldots, d_{h-1}^*)$ maximizing the cumulated discounted reward over h stages

$$R(x_0, d_0, \ldots, d_{h-1}) = \sum_{t=0}^{h-1} \gamma^t r(x_t, d_t).$$

Different kinds of optimal policies:

Open loop:	$d_t^* = d^*(x_0, t)$	(OK in the deterministic case)
Closed loop:	$d_t^* = d^*(x_t, t)$	(Also OK in the stochastic case)
Stationary:	$d_t^* = d^*(x_t)$	(OK in the infinite horizon case)

To learn, let's assume available a sample of N h-stage trajectories

$$(x_0, d_0, r_0, x_1, d_1, r_1, \ldots, x_{h-1}, d_{h-1}, r_{h-1}, x_h)^N$$

Learning a sequential decision policy

We first assume that the decisions shown in the sample are the optimal ones:

Learning open-loop policy: can use open loop parts of sample
$(x_0, d_0^*, \ldots, d_{h-1}^*)^N$
$\Rightarrow$ apply SL h times,$\forall t = 0, \ldots, h-1$, to construct $\hat{d}^*(x_0, t)$ from $(x_0, d_t^*)^N$.

Learning a sequential decision policy

We first assume that the decisions shown in the sample are the optimal ones:

Learning open-loop policy: can use open loop parts of sample
$(x_0, d^*_0, \ldots, d^*_{h-1})^N$
$\Rightarrow$ apply SL h times,$\forall t = 0, \ldots, h-1$, to construct $\hat{d}^*(x_0, t)$ from $(x_0, d^*_t)^N$.

Learning closed-loop policy: should use closed loop parts of sample
$(x_0, x_1, \ldots, x_{h-1}, d^*_0, \ldots, d^*_{h-1})^N$
$\Rightarrow$ apply SL h times, $\forall t = 0, \ldots, h-1$, to construct $\hat{d}^*(x_t, t)$ from $(x_t, d^*_t)^N$.

Learning a sequential decision policy

We first assume that the decisions shown in the sample are the optimal ones:

Learning open-loop policy: can use open loop parts of sample
$(x_0, d^*_0, \ldots, d^*_{h-1})^N$
$\Rightarrow$ apply SL h times,$\forall t = 0, \ldots, h-1$, to construct $\hat{d}^*(x_0, t)$ from $(x_0, d^*_t)^N$.

Learning closed-loop policy: should use closed loop parts of sample
$(x_0, x_1, \ldots, x_{h-1}, d^*_0, \ldots, d^*_{h-1})^N$
$\Rightarrow$ apply SL h times, $\forall t = 0, \ldots, h-1$, to construct $\hat{d}^*(x_t, t)$ from $(x_t, d^*_t)^N$.

Possible discussion: for the stochastic case, the closed-loop approach still holds valid.

Learning a sequential decision policy

Problem: How to generalize this if the decisions shown in the sample are random (i.e. not necessarily the optimal ones)?

Brute force approach: one could use open loop parts of sample $(x_0, d_0, \ldots, d_{h-1}, R)^N$
$\Rightarrow$ apply SL to construct $\hat{R}(x_0, d_0, \ldots, d_{h-1})$
$\Rightarrow$ compute $(d_0, \ldots, d_{h-1})^*(x) = \arg\max_{d_0,\ldots,d_{h-1}} \hat{R}(x_0, d_0, \ldots, d_{h-1})$.

Learning a sequential decision policy

Problem: How to generalize this if the decisions shown in the sample are random (i.e. not necessarily the optimal ones)?

Brute force approach: one could use open loop parts of sample

$(x_0, d_0, \ldots, d_{h-1}, R)^N$

$\Rightarrow$ apply SL to construct $\hat{R}(x_0, d_0, \ldots, d_{h-1})$

$\Rightarrow$ compute $(d_0, \ldots, d_{h-1})^*(x) = \arg\max_{d_0,\ldots,d_{h-1}} \hat{R}(x_0, d_0, \ldots, d_{h-1})$.

Possible discussion: computational complexity of $\arg\max$*; sample complexity; what if system is stochastic...*

Learning a sequential decision policy

Model based approach:

- Exploit sample of state transitions $(x_{t_i}, d_{t_i}, x_{t_i+1})^{N \times h}$
  $\Rightarrow$ use SL to build model of dynamics $\hat{f}(x, d)$.
- Exploit sample of instantaneous rewards $(x_{t_i}, d_{t_i}, r_{t_i})^{N \times h}$
  $\Rightarrow$ use SL to build model of reward function $\hat{r}(x, d)$.
- Use dynamic programming algorithms (e.g. value iteration) to compute from $\hat{f}(x, d)$ and $\hat{r}(x, d)$ the optimal policy $\hat{d}^*(x, t)$.

Learning a sequential decision policy

Model based approach:

- Exploit sample of state transitions $(x_{t_i}, d_{t_i}, x_{t_i+1})^{N \times h}$
  $\Rightarrow$ use SL to build model of dynamics $\hat{f}(x, d)$.
- Exploit sample of instantaneous rewards $(x_{t_i}, d_{t_i}, r_{t_i})^{N \times h}$
  $\Rightarrow$ use SL to build model of reward function $\hat{r}(x, d)$.
- Use dynamic programming algorithms (e.g. value iteration) to compute from $\hat{f}(x, d)$ and $\hat{r}(x, d)$ the optimal policy $\hat{d}^*(x, t)$.

Possible discussion: can be modified to work in the stochastic case, makes good use of sample information; exploits stationary problem structure; has computational complexity of DP, therefore is problematic in continuous or high-dimensional state spaces ...

Learning a sequential decision policy

Proposed approach

Interlacing batch-mode SL and backward value-iteration:

- Assume we are in a certain state x at time $h-1$ (last stage):
  $\Rightarrow$ define $Q_1(x,d) \equiv r(x,d)$
  $\Rightarrow$ optimal decision: $d^*(x,h-1) = \arg\max_d Q_1(x,d)$
  $\Rightarrow$ SL on $(x_{t_i}, d_{t_i}, r_{t_i})^{N\times h}$ to compute $\hat{Q}_1(x,d)$ + arg max...

Learning a sequential decision policy

Proposed approach

Interlacing batch-mode SL and backward value-iteration:

- Assume we are in a certain state x at time $h-1$ (last stage):
  $\Rightarrow$ define $Q_1(x,d) \equiv r(x,d)$
  $\Rightarrow$ optimal decision: $d^*(x,h-1) = \arg\max_d Q_1(x,d)$
  $\Rightarrow$ SL on $(x_{t_i}, d_{t_i}, r_{t_i})^{N\times h}$ to compute $\hat{Q}_1(x,d)$ + arg max...
- Assume we are in state x at time $h-2$:
  $\Rightarrow$ define $Q_2(x,d) = r(x,d) + \gamma \arg\max_{d'} Q_1(f(x,d),d')$
  $\Rightarrow$ optimal decision to take: $d^*(x,h-2) = \arg\max_d Q_2(x,d)$
  $\Rightarrow$ SL on $(x_{t_i}, d_{t_i}, r_{t_i} + \gamma \arg\max_{d'} \hat{Q}_1(x_{t_i+1}, d'))^{N\times h}$
  $\Rightarrow \hat{Q}_2(x,d)$ + arg max $\Rightarrow \hat{d}^*(x,h-2)$

Learning a sequential decision policy

Proposed approach

Interlacing batch-mode SL and backward value-iteration:

- Assume we are in a certain state x at time $h-1$ (last stage):
  $\Rightarrow$ define $Q_1(x,d) \equiv r(x,d)$
  $\Rightarrow$ optimal decision: $d^*(x,h-1) = \arg\max_d Q_1(x,d)$
  $\Rightarrow$ SL on $(x_{t_i}, d_{t_i}, r_{t_i})^{N\times h}$ to compute $\hat{Q}_1(x,d)$ + arg max...
- Assume we are in state x at time $h-2$:
  $\Rightarrow$ define $Q_2(x,d) = r(x,d) + \gamma \arg\max_{d'} Q_1(f(x,d),d')$
  $\Rightarrow$ optimal decision to take: $d^*(x,h-2) = \arg\max_d Q_2(x,d)$
  $\Rightarrow$ SL on $(x_{t_i}, d_{t_i}, r_{t_i} + \gamma \arg\max_{d'} \hat{Q}_1(x_{t_i+1}, d'))^{N\times h}$
  $\Rightarrow$ $\hat{Q}_2(x,d)$ + arg max $\Rightarrow$ $\hat{d}^*(x,h-2)$
- Continue $h-2$ further times to yield sequence of $\hat{Q}_i$-functions and policy approximations $\hat{d}^*(x,h-i)$, $\forall t = 1,\ldots,h$.

Learning a sequential decision policy

Proposed approach

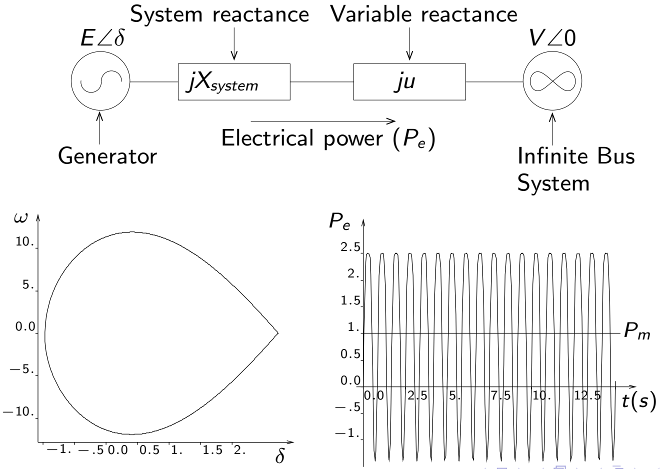
Interlacing batch-mode SL and backward value-iteration:

- Assume we are in a certain state x at time $h-1$ (last stage):
  $\Rightarrow$ define $Q_1(x,d) \equiv r(x,d)$
  $\Rightarrow$ optimal decision: $d^*(x,h-1) = \arg\max_d Q_1(x,d)$
  $\Rightarrow$ SL on $(x_{t_i}, d_{t_i}, r_{t_i})^{N\times h}$ to compute $\hat{Q}_1(x,d)$ + arg max...
- Assume we are in state x at time $h-2$:
  $\Rightarrow$ define $Q_2(x,d) = r(x,d) + \gamma \arg\max_{d'} Q_1(f(x,d),d')$
  $\Rightarrow$ optimal decision to take: $d^*(x,h-2) = \arg\max_d Q_2(x,d)$
  $\Rightarrow$ SL on $(x_{t_i}, d_{t_i}, r_{t_i} + \gamma \arg\max_{d'} \hat{Q}_1(x_{t_i+1}, d'))^{N\times h}$
  $\Rightarrow$ $\hat{Q}_2(x,d)$ + arg max $\Rightarrow$ $\hat{d}^*(x,h-2)$
- Continue $h-2$ further times to yield sequence of $\hat{Q}_i$-functions and policy approximations $\hat{d}^*(x,h-i)$, $\forall t = 1,\ldots,h$.
- This algorithm is called "Fitted Q iteration".

Fitted Q iteration: discussion

- All what the algorithm actually needs to work is a sample of four-tuples $(x_{t_i}, d_{t_i}, r_{t_i}, x_{t_i+1})^N$, and a a good supervised learning algorithm (for least squares regression).
- It has be shown to give excellent results when using ensemble of regression trees as supervised learning algorithms
- It can cope (without modification) with stochastic problems.
- It can exploit efficiently very large samples.
- It already proved to work well on several complex continuous state space problems.
- Iterating it sufficiently many times, it yields an approximation of the optimal (closed-loop, and stationary) infinite horizon decision policy.

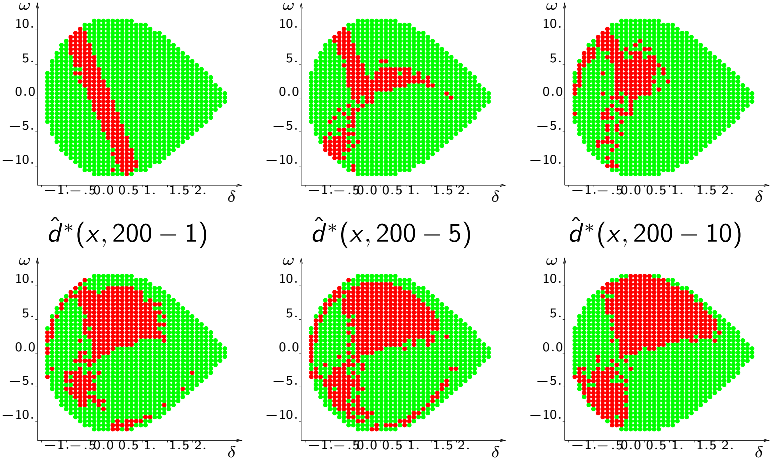
Illustration: power system control



The sequential decision problem

- Two state variables: δ and ω.
- Time discretization: time between t and $t+1$ equal to 50 ms
- Two possible decisions d: capacitance set to zero or capacitance set to its maximal value.
- $r(x_t, u_t, w_t) = -|P_{e\,t+1} - P_m|$ if $x_{t+1} \in$ *stability domain* and -100 otherwise
- $\gamma = 0.98$
- Sequential decision problem such that the optimal stationary policy damps the electrical power oscillations

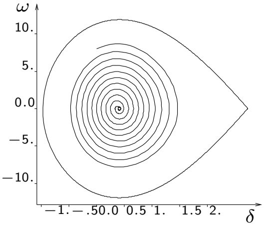
Representation of $\hat{d}^*(x, t)$ ($h = 200$), 10,000 four-tuples

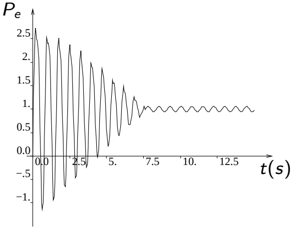


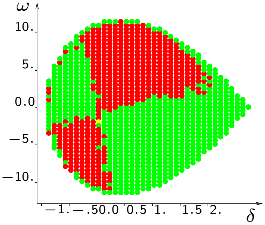
$\hat{d}^*(x, 200-1)$ $\hat{d}^*(x, 200-5)$ $\hat{d}^*(x, 200-10)$

$\hat{d}^*(x, 200-20)$ $\hat{d}^*(x, 200-50)$ $\hat{d}^*(x, 0)$

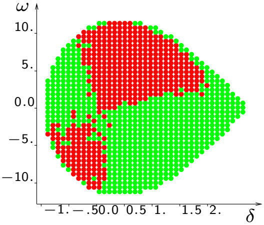
$\hat{d}^*(x) = \hat{d}^*(x, 0)$
used to control
the system
($h = 200$)
10,000 samples



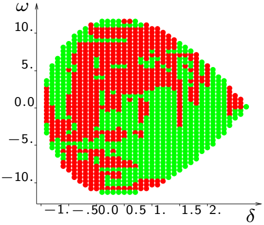




$\hat{d}^*(x, 0)$
100,000 samples



$\hat{d}^*(x, 0)$
10,000 samples



$\hat{d}^*(x, 0)$
1000 samples

Finish

- Fitted Q iteration algorithm combined with ensemble of regression trees has been evaluated on several problems and was constantly giving second to none performances.
- Why has not this algorithm been proposed before ?