# A Series of Lectures on Approximate Dynamic Programming Lecture 2

Dimitri P. Bertsekas

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology

University of Cyprus
September 2017

# APPROXIMATE DYNAMIC PROGRAMMING I

# Outline

# Recall our Problem Structure

## Discrete-time system

$$x_{k+1} = f_k(x_k, u_k, w_k), \qquad k = 0, 1, \ldots, N-1$$

- $x_k$: State
- $u_k$: Control from a constraint set $U_k(x_k)$
- $w_k$: Disturbance; random parameter with distribution $P(w_k \mid x_k, u_k)$

## Optimization over Feedback Policies $\pi = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$, with $u_k = \mu_k(x_k) \in U(x_k)$

Cost of a policy starting at initial state $x_0$:

$$J_\pi(x_0) = E\left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k\big(x_k, \mu_k(x_k), w_k\big) \right\}$$

Optimal cost function:

$$J^*(x_0) = \min_\pi J_\pi(x_0)$$

# Recall the Exact DP Algorithm

Computes for all $k$ and states $x_k$: $J_k(x_k)$, the opt. cost of tail problem that starts at $x_k$

**Go backwards, $k = N-1, \ldots, 0$, using**

$$J_N(x_N) = g_N(x_N)$$

$$J_k(x_k) = \min_{u_k \in U_k(x_k)} E\Big\{g_k(x_k, u_k, w_k) + J_{k+1}\big(f_k(x_k, u_k, w_k)\big)\Big\}$$

**Notes:**

- $J_0(x_0) = J^*(x_0)$: Cost generated at the last step, is equal to the optimal cost
- Let $\mu_k^*(x_k)$ minimize in the right side above for each $x_k$ and $k$. Then the policy $\pi^* = \{\mu_0^*, \ldots, \mu_{N-1}^*\}$ is optimal

# Practical Difficulties of DP

## The curse of dimensionality (too many values of $x_k$)

- In continuous-state problems:
  - Discretization needed
  - Exponential growth of the computation with the dimensions of the state and control spaces
- In naturally discrete/combinatorial problems: Quick explosion of the number of states as the search space increases
- Length of the horizon (what if it is infinite?)

## The curse of modeling; we may not know exactly $f_k$ and $P(x_k \mid x_k, u_k)$

- It is often hard to construct an accurate math model of the problem
- Sometimes a simulator of the system is easier to construct than a model

## The problem data may not be known well in advance

- A family of problems may be addressed. The data of the problem to be solved is given with little advance notice
- The problem data may change as the system is controlled – need on-line replanning and fast solution

# A MAJOR IDEA: Cost Approximation

## One-Step Lookahead - Idea is to simplify the DP computation

- Replace $J_{k+1}$ by an approximation $\tilde{J}_{k+1}$
- Apply $\bar{u}_k$ that attains the minimum in

$$\min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big) \Big\}$$

## $\ell$-Step Lookahead

- At state $x_k$ solve the $\ell$-step DP problem starting at $x_k$ and using terminal cost $\tilde{J}_{k+\ell}$
- If $\overline{u}_k, \overline{\mu}_{k+1}, \ldots, \overline{\mu}_{k+\ell-1}$ is an optimal policy for the $\ell$-step problem, apply the first control $\overline{u}_k$

## Other Names Used

Rolling or receding horizon control

**Let's focus on the one-step lookahead computation at stage $k$**

$$\min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big) \Big\}$$

**Issues**

- A key issue: How do we choose the approximate cost functions $\tilde{J}_k$?
- Another issue: How do we deal with the minimization and the computation of $E\{\cdot\}$

**A variety of approximation approaches (and combinations thereoff):**

- Parametric cost-to-go approximation: Use as $\tilde{J}_k$ a parametric function $\tilde{J}_k(x_k, r_k)$ (e.g., a neural network), whose parameter $r_k$ is "tuned" by some scheme
- Rollout: Use as $\tilde{J}_k$ the cost of some suboptimal policy, which is calculated either analytically or by simulation
- Problem approximation: Use $\tilde{J}_k$ derived from a related but simpler problem

**At State $x_k$**

**DP minimization**

First $\ell$ Steps "Future"

$$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{ g_k(x_k,u_k,w_k) + \sum_{m=k+1}^{k+\ell-1} g_k\big(x_m,\mu_m(x_m),w_m\big) + \tilde J_{k+\ell}(x_{k+\ell}) \right\}$$

**Lookahead Minimization**

**Cost-to-go Approximation**

**At State $x_k$**

**DP minimization**

$$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{ \overbrace{g_k(x_k,u_k,w_k) + \sum_{m=k+1}^{k+\ell-1} g_k\big(x_m,\mu_m(x_m),w_m\big)}^{\text{First } \ell \text{ Steps}} + \overbrace{\tilde{J}_{k+\ell}(x_{k+\ell})}^{\text{“Future”}} \right\}$$

**Computation of $\tilde{J}_{k+\ell}$:**

Simple choices

Parametric approximation

Rollout

Tail problem approximation

# Approximate Minimization Possibilities

**At State $x_k$**

**DP minimization**

First $\ell$ Steps — "Future"

$$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{ g_k(x_k,u_k,w_k) + \sum_{m=k+1}^{k+\ell-1} g_k\big(x_m,\mu_m(x_m),w_m\big) + \tilde{J}_{k+\ell}(x_{k+\ell}) \right\}$$

**Approximations:**

Replace $E\{\cdot\}$ with nominal values (certainty equivalent control)

Limited simulation (Monte Carlo tree search)

**Computation of $\tilde{J}_{k+\ell}$:**

Simple choices

Parametric approximation

Rollout

Tail problem approximation

# A First-Order Division of Lookahead Choices

## Long lookahead $\ell$ and simple choice of $\tilde{J}_{k+\ell}$

- Some examples

$$\tilde{J}_{k+\ell}(x) \equiv 0 \qquad \text{(or a constant)}$$

$$\tilde{J}_{k+\ell}(x) = g_N(x)$$

For problems with a "goal state" use a simple penalty $\tilde{J}_{k+\ell}$

$$\tilde{J}_{k+\ell}(x) = \begin{cases} 0 & \text{if } x \text{ is a goal state} \\ >> 1 & \text{if } x \text{ is not a goal state} \end{cases}$$

- Long lookahead $\Longrightarrow$ A lot of DP computation
- Often must be done off-line

## Short lookahead $\ell$ and sophisticated choice $\tilde{J}_{k+\ell} \approx J_{k+\ell}$

- The lookahead cost function approximates (to within a constant) the optimal cost-to-go produced by exact DP
- We will next describe a variety of off-line and on-line approximation approaches

**Lookahead Minimization**

**Cost-to-go Approximation**

First $\ell$ Steps

"Future"

$$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{g_k(x_k,u_k,w_k)+\sum_{m=k+1}^{k+\ell-1} g_k\big(x_m,\mu_m(x_m),w_m\big)+\tilde J_{k+\ell}(x_{k+\ell})\right\}$$

Parametric approximation

# Parametric Approximation Architectures

$$J_k(x_k) \approx \tilde{J}_k(x_k, r_k)$$

with

$$r_k = (r_{1,k}, \ldots, r_{m,k}) \text{ a vector of “tunable"scalar weights}$$

- We use $\tilde{J}_k$ in place of $J_k$ (the optimal cost-to-go function) in a one-step or multistep lookahead scheme
- $\tilde{J}_k(x_k, r_k)$ is called an approximation architecture
- Role of $r_k$: By adjusting $r_k$ we can change the “shape" of $\tilde{J}_k$ so that it is “close" to to the optimal $J_k$ (at least within a constant)

## Two key Issues

- The choice of the parametric class $\tilde{J}_k(x_k, r_k)$; there is a large variety
- The method for tuning/adjusting the weights (“training" the architecture)

# Feature-Based Architectures

## Feature extraction

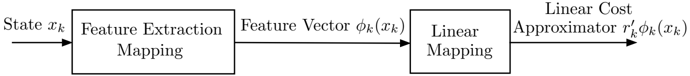
- A process that maps the state $x_k$ into a vector $\phi_k(x_k) = \big(\phi_{1,k}(x_k), \ldots, \phi_{m,k}(x_k)\big)$, called the feature vector associated with $x_k$
- A feature-based cost approximator has the form

$$\tilde{J}_k(x_k, r_k) = \hat{J}_k\big(\phi_k(x_k), r_k\big)$$

where $r_k$ is a parameter vector and $\hat{J}_k$ is some function, linear or nonlinear in $r_k$

- With a well-chosen feature vector $\phi_k(x_k)$, a good approximation to the cost-to-go is often provided by linearly weighting the features, i.e.,

$$\tilde{J}_k(x_k, r_k) = \hat{J}_k\big(\phi_k(x_k), r_k\big) = \sum_{i=1}^{m} r_{i,k}\phi_{i,k}(x_k) = r_k'\phi_k(x_k)$$

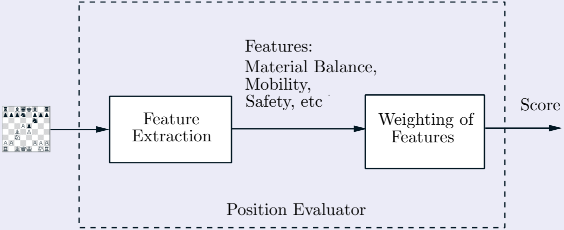



This can be viewed as subspace approximation; view the features the $\phi_{i,k}(x_k)$ as basis functions

# Feature Selection: A Major Issue

- Any generic basis functions, such as classes of polynomials, wavelets, radial basis functions, etc, can serve as features
- In some cases, problem-specific features can be "hand-crafted"

## Computer chess example




- Think of state: board position; control: move choice
- Use a feature-based position evaluator assigning a score to each position
- Most chess programs use a linear architecture with "manual" choice of weights
- Some computer programs choose the weights by a least squares fit using lots of grandmaster play examples

# An Example of Architecture Training: Sequential DP Approximation

## A common way to train architectures $\tilde{J}_k(x_k, r_k)$ in the context of DP

- We start with $\tilde{J}_N = g_N$ and sequentially train going backwards, until $k = 1$
- Given a cost-to-go approximation $\tilde{J}_{k+1}$, we use one-step lookahead to construct a large number of state-cost pairs $(x_k^s, \beta_k^s)$, $s = 1, \ldots, q$, where

$$\beta_k^s = \min_{u \in U_k(x_k^s)} E\Big\{ g(x_k^s, u, w_k) + \tilde{J}_{k+1}\big(f_k(x_k^s, u, w_k), r_{k+1}\big) \Big\}, \qquad s = 1, \ldots, q$$

- We "train" an architecture $\tilde{J}_k$ on the training set $(x_k^s, \beta_k^s)$, $s = 1, \ldots, q$

## Training by least squares/regression

- We minimize over $r_k$

$$\sum_{s=1}^{q} \big( \tilde{J}_k(x_k^s, r_k) - \beta^s \big)^2 + \gamma \| r_k - \bar{r} \|^2$$

  where $\bar{r}$ is an initial guess for $r_k$ and $\gamma > 0$ is a regularization parameter
- Incremental gradient methods are typically used for this. They take advantage of the large sum structure of the cost function
- For a linear architecture the training problem is a linear least squares problem

# Neural Networks for Parametric Approximation

Neural nets can be used in a sequential DP approximation scheme: Train the stage $k$ neural net (i.e., compute $\tilde{J}_k$) using a training set generated with the stage $k+1$ neural net (which defines $\tilde{J}_{k+1}$)

## Focus at the typical stage $k$ and drop the index $k$ for convenience

- Neural nets are approximation architectures of the form

$$\tilde{J}(x, v, r) = \sum_{i=1}^{m} r_i \phi_i(x, v) = r' \phi(x, v)$$

  involving two parameter vectors $r$ and $v$ with different roles
- View $\phi(x, v)$ as a feature vector; view $r$ as a vector of linear weighting parameters for $\phi(x, v)$
- The training is done by least squares/regression
- By training $v$ jointly with $r$, we obtain automatically generated features!