A Series of Lectures on Approximate Dynamic Programming Lecture 3

Dimitri P. Bertsekas

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology

University of Cyprus
September 2017

APPROXIMATE DYNAMIC PROGRAMMING II

Outline

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

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

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

One-Step and Multistep Lookahead

One-Step Lookahead

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

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

ℓ-Step Lookahead

- At state x_k solve the ℓ-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 ℓ-step problem, apply the first control $\overline{u}_k$

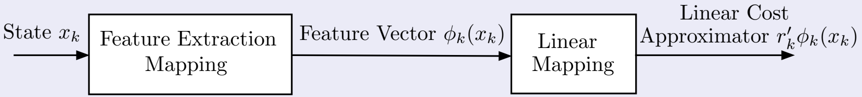
Lookahead Minimization — **Cost-to-go Approximation**

First ℓ 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

Feature-based architectures: The linear case



Architecture Training by Sequential DP Approximation

- Start with $\tilde{J}_N = g_N$ and sequentially train going backwards, until $k = 0$
- 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, β_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, β_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

Neural Networks for Constructing Cost-to-Go Approximations $\tilde{J}_k$

Neural nets can be used in the 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 $\tilde{J}_{k+1}$)

Two ways to view neural networks

- As nonlinear approximation architectures
- As linear architectures with automatically constructed features

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

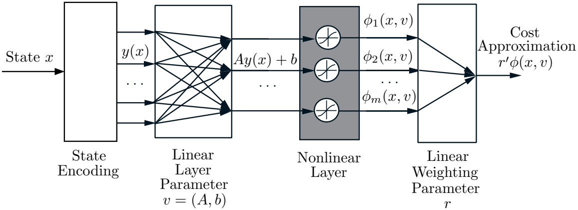
- Neural nets are approximation architectures of the form

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

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 $\phi(x, v)$
- By training v jointly with r, we obtain automatically generated features!

Neural Network with a Single Nonlinear Layer

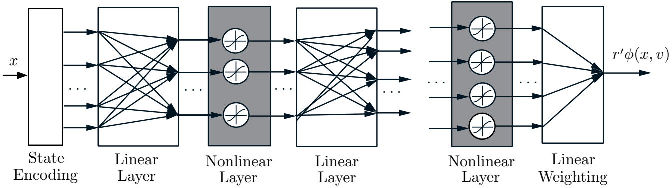


- State encoding (could be the identity, could include special features of the state)
- Linear layer $Ay(x)+b$ [parameters to be determined: $v=(A,b)$]
- Nonlinear layer produces m outputs $\phi_i(x,v)=\sigma\Big(\big(Ay(x)+b\big)_i\Big)$, $i=1,\ldots,m$
- σ is a scalar nonlinear differentiable function; several types have been used (hyperbolic tangent, logistic, rectified linear unit)
- Training problem is to use the training set (x^s,β^s), $s=1,\ldots,q$, for

$$\min_{A,b,r}\sum_{s=1}^{q}\left(\sum_{i=1}^{m} r_i\,\sigma\left(\big(Ay(x^s)+b\big)_i\right)-\beta^s\right)^2+\text{(Regularization Term)}$$

- Solved often with incremental gradient methods (known as backpropagation)
- Universal approximation theorem: With sufficiently large number of parameters, “arbitrarily" complex functions can be closely approximated

Deep Neural Networks



- More complex NNs are formed by concatenation of multiple layers
- The outputs of each nonlinear layer become the inputs of the next linear layer
- Considerable success has been achieved in major contexts

Possible reasons for the success

- The multilayer network provides a hierarchy of features (each set of features being a function of the preceding set of features) that can be exploited to specialize the role of some of the layers
- We may use matrices A with a special structure that encodes special linear operations such as convolution
- When such structures are used, the training problem may become easier, because the number of parameters in the linear layers is drastically decreased

Q-Factors

- The Q-factor of a state-control pair (x_k, u_k) at time k is defined by

$$Q_k(x_k, u_k) = E\Big\{g_k(x_k, u_k, w_k) + J_{k+1}(x_{k+1})\Big\}$$

where J_{k+1} is the optimal cost-to-go function for stage $k+1$

- Note that

$$J_k(x_k) = \min_{u \in U_k(x_k)} Q_k(x_k, u_k)$$

so the DP algorithm is written in terms of Q_k

$$Q_k(x_k, u_k) = E\Big\{g_k(x_k, u_k, w_k) + \min_{u \in U_{k+1}(x_{k+1})} Q_{k+1}(x_{k+1}, u)\Big\}$$

- We approximate this algorithm using a Q-factor approximation architecture $\tilde{Q}_k(x_k, u_k, r_k)$

$$\tilde{Q}_k(x_k, u_k, r_k) = E\Big\{g_k(x_k, u_k, w_k) + \min_{u \in U_{k+1}(x_{k+1})} \tilde{Q}_{k+1}(x_{k+1}, u, r_{k+1})\Big\}$$

Approximation in Q-Factor Space: Using a Simulator Instead of a Model

- Consider sequential DP approximation of Q-factor parametric approximations

$$\tilde{Q}_k(x_k, u_k, r_k) = E\Big\{g_k(x_k, u_k, w_k) + \min_{u \in U_{k+1}(x_{k+1})} \tilde{Q}_{k+1}(x_{k+1}, u, r_{k+1})\Big\}$$

  (Note a mathematical magic: The order of $E\{\cdot\}$ and min have been reversed.)

- We obtain $\tilde{Q}_k(x_k, u_k, r_k)$ by training with many pairs $\big((x_k^s, u_k^s), \beta_k^s\big)$, where β_k^s is a sample of the approximate Q-factor of (x_k^s, u_k^s). [No need to compute $E\{\cdot\}$]
- Note: No need for a model to obtain β_k^s. Sufficient to have a simulator that generates state-control-cost-next state random samples

$$\big((x_k, u_k), (g_k(x_k, u_k, w_k), x_{k+1})\big)$$

- Having computed r_k, the one-step lookahead control is obtained on-line as

$$\overline{\mu}_k(x_k) = \arg\min_{u \in U_k(x_k)} \tilde{Q}_k(x_k, u, r_k)$$

  without the need of a model or expected value calculations
- Thus the on-line calculation of the control is simplified

Lookahead Minimization

Cost-to-go Approximation

First ℓ Steps

"Future"

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

Monte Carlo tree search

Rollout: Simulation with fixed policy

Parametric approximation at the end

Rollout: A General Method to Compute Cost-to-Go Approximations

Computes the lookahead functions $\tilde{J}_k$ as the cost-to-go functions of some suboptimal policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, referred to as the base policy or base heuristic

Rollout implementation

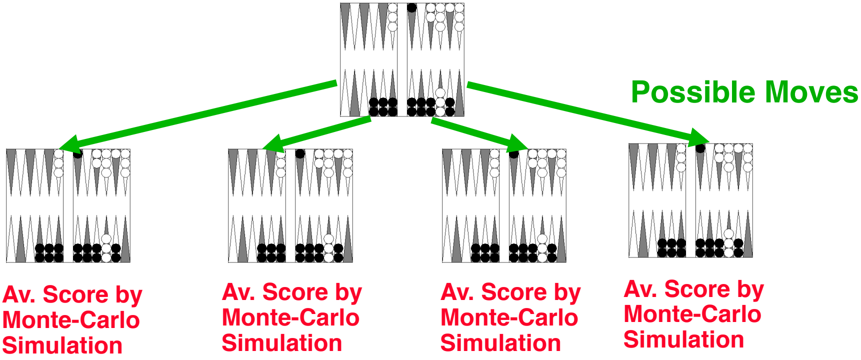
- We may use rollout in one-step or multistep lookahead
- We may calculate the base policy costs $\tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big)$ needed in

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

  (or its multistep version) analytically or by simulation
- A variant: The base policy costs $\tilde{J}_{k+1}$ may be approximated over a limited rolling horizon, with a terminal cost approximation added at the end
- Simulation may be used for calculation of needed values of $\tilde{J}_{k+1}$ on-line
- The amount of simulation needed may be overwhelming (parallel computation helps). Simulation greatly simplifies if the problem is deterministic

Major fact about rollout

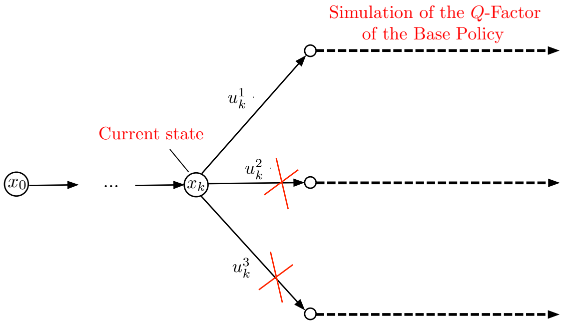
The rollout policy performs at least as well as the base policy. The improvement is often DRAMATIC. Relation to policy iteration method of infinite horizon DP



The original player (Tesauro, 1996):

- Involved one-step lookahead
- Base heuristic was a (relatively crude) backgammon player developed by different approximate DP methods
- The program played competitively against the best humans
- Was very time consuming (lots of parallelization of MC simulation)

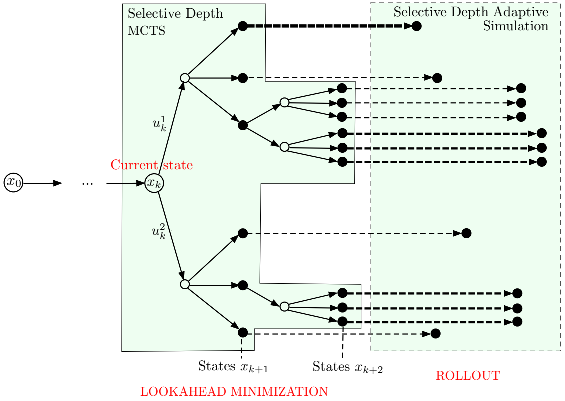
Stochastic Rollout with Adaptive Simulation



Adaptive simulation aims to reduce the simulation effort

- Based on simulation results, we may discard some of the controls u_k that are “clearly" inferior
- For this we may use statistical tests (“confidence intervals")
- The idea can be extended to multistep lookahead
- In some variants the rollout may include a limited horizon and cost function approximation

Stochastic Rollout with Monte Carlo Tree Search



MCTS aims to combine rollout simulation and lookahead minimization

- Motivation: Some controls u_k that appear to be promising, may be worth exploring better through multistep lookahead
- MCTS combines selective depth lookahead and adaptive simulation

Example of Rollout + Terminal Cost Approximation: AlphaGo

Recent success: A Go program that plays at the level of the best humans

- Combines many of the ideas that we have discussed with awesome computing power and many heuristics
- Multistep lookahead with Monte Carlo tree search
- Rollout with rolling horizon and cost function approximation (computed off-line with deep neural network)
- The base policy of the rollout is also computed off-line
- Massive on-line computation: 1920 CPUs and 280 GPUs, $3000 electric bill per game!