Reinforcement Learning and Optimal Control

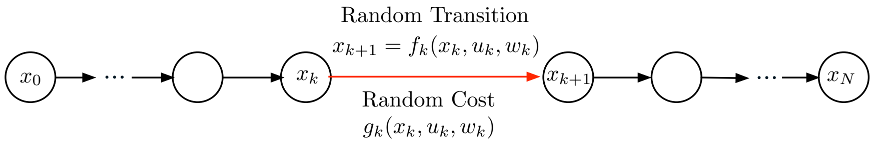
ASU, CSE 691, Winter 2019

Dimitri P. Bertsekas
dimitrib@mit.edu

Lecture 13
A Review of the Course

Outline

Stochastic Finite Horizon DP Problems



- System $x_{k+1} = f_k(x_k, u_k, w_k)$ with state x_k, control u_k, and random "disturbance" w_k
- Cost function:

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

- Policies $\pi = \{\mu_0, \dots, \mu_{N-1}\}$, where μ_k is a "closed-loop control law" or "feedback policy"/a function of x_k. Specifies control $u_k = \mu_k(x_k)$ to apply when at x_k.
- For given initial state x_0, minimize over all $\pi = \{\mu_0, \dots, \mu_{N-1}\}$ the cost

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

- Optimal cost function $J^*(x_0) = \min_\pi J_\pi(x_0)$

The Stochastic DP Algorithm

Produces the optimal costs $J_k^*(x_k)$ of the tail subproblems that start at x_k

Start with $J_N^*(x_N) = g_N(x_N)$, and for $k = 0, \ldots, N-1$, let

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big)\Big\}, \qquad \text{for all } x_k.$$

- The optimal cost $J^*(x_0)$ is obtained at the last step: $J_0^*(x_0) = J^*(x_0)$.

On-line implementation of the optimal policy, given $J_1^*, \ldots, J_{N-1}^*$

Sequentially, going forward, for $k = 0, 1, \ldots, N-1$, observe x_k and apply

$$u_k^* \in \arg \min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big)\Big\}.$$

Issues: Need to compute J_{k+1}^* (possibly off-line), compute expectation for each u_k, minimize over all u_k

Approximation in value space: Use $\tilde{J}_{k+1}$ in place of J_{k+1}^*; also approximate $E\{\cdot\}$ and $\min_{u_k}$.

Approximation in Value Space

Approximate Min
Discretization

$$\min_{u_k} E\Big\{\overbrace{g_k(x_k,u_k,w_k)}^{\text{First Step}}+\overbrace{\tilde J_{k+1}(x_{k+1})}^{\text{“Future”}}\Big\}$$

Approximate $E\{\cdot\}$

Certainty equivalence
Adaptive simulation
Monte Carlo tree search

Approximate Cost-to-Go $\tilde J_{k+1}$

Problem approximation
Rollout, Model Predictive Control
Parametric approximation
Neural nets
Aggregation

ONE-STEP LOOKAHEAD

At State x_k
DP minimization

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

Lookahead Minimization

Cost-to-go Approximation

MULTISTEP LOOKAHEAD

Approximation in Policy Space: The Major Alternative to Approximation in Value Space

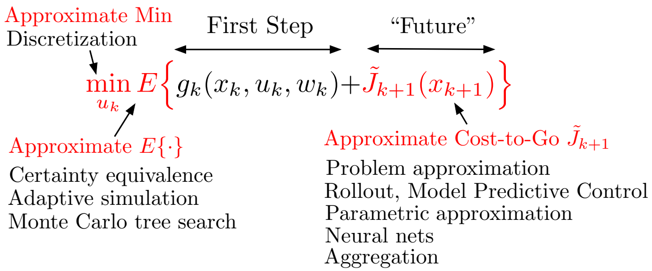
- Idea: Select the policy by optimization over a suitably restricted class of policies.
- The restricted class is usually a parametric family of policies $\mu_k(x_k, r_k)$, $k = 0, \ldots, N-1$, of some form, where r_k is a parameter (e.g., a neural net).
- Important advantage once the parameter r_k is computed: The computation of controls during on-line operation of the system is often much easier: At state x_k apply $u_k = \mu_k(x_k, r_k)$.

Approximation in policy space on top of approximation in value space

- Compute approximate cost-to-go functions $\tilde{J}_{k+1}$, $k = 0, \ldots, N-1$.
- This defines the corresponding suboptimal policy $\tilde{\mu}_k$, $k = 0, \ldots, N-1$, through one-step or multistep lookahead.
- Approximate $\tilde{\mu}_k$ using some form of regression and a training set consisting of a large number q of sample pairs $\left(x_k^s, u_k^s\right)$, $s = 1, \ldots, q$, where $u_k^s = \tilde{\mu}_k(x_k^s)$.
- Example: Introduce a parametric family of policies $\mu_k(x_k, r_k)$, $k = 0, \ldots, N-1$, of some form, where r_k is a parameter. Then estimate the parameters r_k by

$$r_k \in \arg\min_r \sum_{s=1}^{q} \left\| u_k^s - \mu_k(x_k^s, r) \right\|^2.$$

On-Line and Off-Line Lookahead Implementations



- Off-line methods: All the functions $\tilde{J}_{k+1}$ are computed for every k, before the control process begins.
- Examples of off-line methods: Neural network and other parametric approximations; also aggregation.
- For many-state problems, the minimizing controls $\tilde{\mu}_k(x_k)$ are computed on-line (because of the storage issue, as well as an off-line excessive computation issue).
- On-line methods: The values $\tilde{J}_{k+1}(x_{k+1})$ are computed only at the relevant next states x_{k+1}, and are used to compute the control to be applied at the N time steps.
- Examples of on-line methods: Rollout and model predictive control.
- On-line methods are well-suited for on-line replanning.

Model-Based Versus Model-Free Implementation

Our layman's use of the term "model-free": A method is called model-free if it involves calculations of expected values using Monte Carlo simulation.

Model-free implementation is necessary when:

- A mathematical model of the probabilities $p_k(w_k \mid x_k, u_k)$ is not available but a computer model/simulator is. For any (x_k, u_k), it simulates sample probabilistic transitions to a successor state x_{k+1}, and generates the corresponding transition costs.
- When for reasons of computational efficiency we prefer to compute the expected value by using sampling and Monte Carlo simulation; e.g., approximate an integral or a huge sum of numbers by a Monte Carlo estimate.

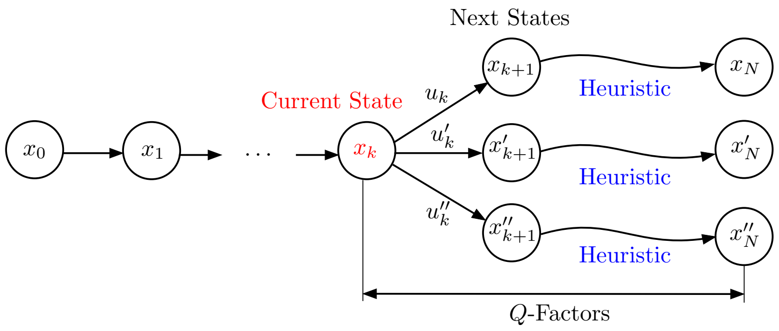
An important example of model-free implementation:

Calculations of approximate Q-factors in lookahead schemes - Approximation in policy space on top of approximation in value space

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

(assuming $\tilde{J}_{k+1}$ has been computed).

General Structure of Deterministic Rollout

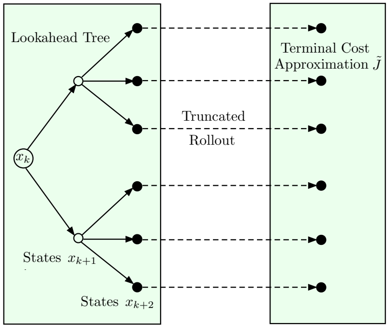


- At state x_k, for every pair (x_k, u_k), $u_k \in U_k(x_k)$, we generate a Q-factor

$$\tilde{Q}_k(x_k, u_k) = g_k(x_k, u_k) + H_{k+1}\big(f_k(x_k, u_k)\big)$$

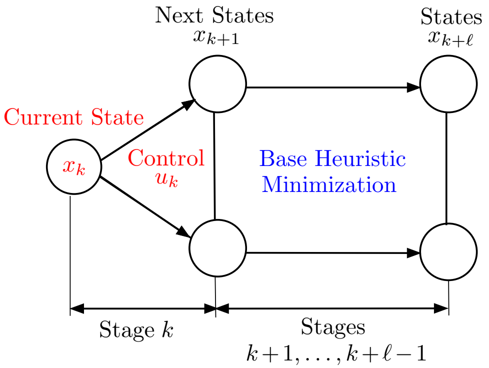
  using the base heuristic [$H_{k+1}(x_{k+1})$ is the heuristic cost starting from x_{k+1}].
- We select the control u_k with minimal Q-factor.
- We move to the next state x_{k+1}, and continue.
- Multistep lookahead versions (length of lookahead is limited by the branching factor of the lookahead tree).

Multistep Lookahead-Truncated Rollout-Terminal Cost Approximation



- Long rollout is costly. It is not necessarily true that increasing the length of the rollout leads to improved performance.
- Terminal cost approximation allows combinations with other value space schemes.
- We can prove cost improvement, assuming various sequential consistency and/or sequential improvement conditions, as well as modifications (fortified rollout).
- Rollout is the most reliable and most easily implementable RL algorithm. Still some trial and error experimentation is recommended for its implementation.

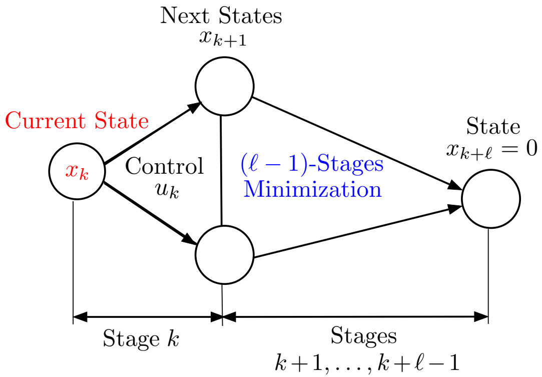
On-Line Rollout for Deterministic Infinite-Spaces Problems



When the control space is infinite rollout needs a different implementation

- One possibility is discretization of $U_k(x_k)$; but then excessive number of Q-factors.
- The major alternative is to use optimization heuristics.
- Seemlessly combine the kth stage minimization and the optimization heuristic into a single ℓ-stage deterministic optimization.
- Can solve it by nonlinear programming/optimal control methods (e.g., quadratic programming, gradient-based).

Model Predictive Control for Deterministic Regulation Problems



- System: $x_{k+1}=f_k(x_k,u_k)$.
- Cost per stage: $g_k(x_k,u_k)\geq 0$, the origin 0 is cost-free and absorbing.
- State and control constraints: $x_k\in X_k$, $u_k\in U_k(x_k)$ for all k.
- At x_k solve an ℓ-step lookahead version of the problem, requiring $x_{k+\ell}=0$ while satisfying the state and control constraints.
- If $\{\tilde{u}_k,\dots,\tilde{u}_{k+\ell-1}\}$ is the control sequence so obtained, apply $\tilde{u}_k$.

Parametric Approximation in Value Space

Approximation Architectures

- A class of functions $\tilde{J}(x, r)$ that depend on x and a vector $r = (r_1, \ldots, r_m)$ of m “tunable" scalar parameters (or weights).
- We adjust r to change $\tilde{J}$ and “match" the cost function approximated.
- Training the architecture: The algorithm to choose r (typically use data/regression).
- Architectures are linear or nonlinear, depending on whether $\tilde{J}(x, r)$ is linear or nonlinear in r.
- Architectures are feature-based if they depend on x via a feature vector $\phi(x)$,

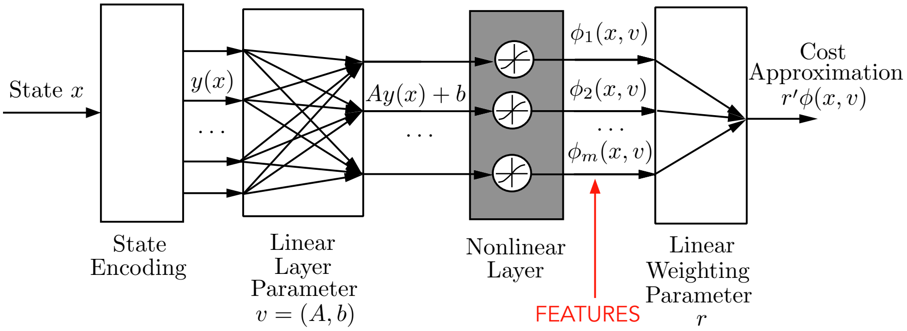
$$\tilde{J}(x, r) = \hat{J}\big(\phi(x), r\big),$$

  where $\hat{J}$ is some function. Idea: Features capture dominant nonlinearities.
- A linear feature-based architecture:

$$\tilde{J}(x, r) = \sum_{\ell=1}^{m} r_\ell \phi_\ell(x),$$

  where r_ℓ and $\phi_\ell(x)$ are the ℓth components of r and $\phi(x)$.

Neural Nets: An Architecture that does not Require Knowledge of Features



- Can be used when problem-specific handcrafted features and linear feature-based architectures are inadequate.
- Tricky training issues by incremental gradient (backpropagation) methods.
- Deep neural nets have proved useful in important contexts.
- There are other nonlinear architectures (e.g., radial basis functions) that we have not covered.

Sequential DP Approximation - A Parametric Approximation at Every Stage (Also Called Fitted Value Iteration)

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), $s = 1, \dots, q$, where

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

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

Typical approach: Train by least squares/regression and possibly using a neural net

We minimize over r_k

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

(plus a regularization term).

Sequential Q-Factor Approximation

- 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: $E\{\min(\dots)\}$ can be sampled; $\min(E\{\dots\})$ cannot be sampled.
- 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 $E\{\cdot\}$.]
- 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

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

without the need of a model or expected value calculations.

Infinite Horizon Finite Spaces Discounted Problems

Convergence of VI

Given any initial conditions $J_0(1), \dots, J_0(n)$, the sequence $\{J_k(i)\}$ generated by VI

$$J_{k+1}(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J_k(j)\big), \qquad i = 1, \dots, n,$$

converges to $J^*(i)$ for each i.

Bellman's equation

The optimal cost function $J^* = \big(J^*(1), \dots, J^*(n)\big)$ satisfies the equation

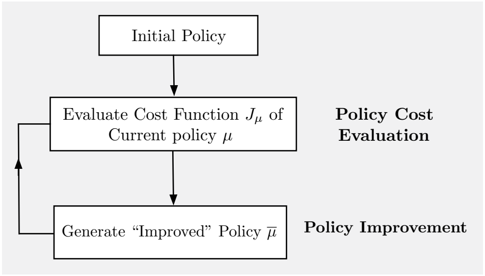
$$J^*(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J^*(j)\big), \qquad i = 1, \dots, n,$$

and is the unique solution of this equation.

Optimality condition

A stationary policy μ is optimal if and only if for every state i, $\mu(i)$ attains the minimum in the Bellman equation.

Policy Iteration (PI) Algorithm


Initial Policy
Evaluate Cost Function J_μ of Current policy μ
Policy Cost Evaluation
Generate "Improved" Policy $\overline{\mu}$
Policy Improvement

Given the current policy μ^k, a PI consists of two phases:

- Policy evaluation computes $J_{\mu^k}(i)$, $i = 1, \dots, n$, as the solution of the (linear) Bellman equation system

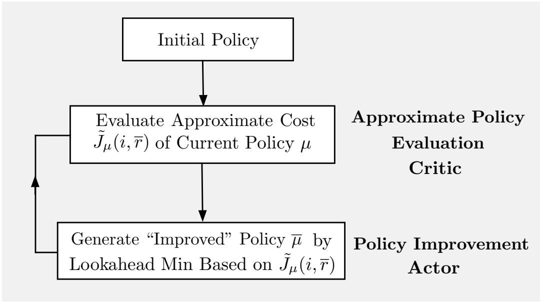
$$J_{\mu^k}(i) = \sum_{j=1}^{n} p_{ij}\big(\mu^k(i)\big)\Big(g\big(i, \mu^k(i), j\big) + \alpha J_{\mu^k}(j)\Big), \quad i = 1, \dots, n$$

- Policy improvement then computes a new policy μ^{k+1} as

$$\mu^{k+1}(i) \in \arg\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i, u, j) + \alpha J_{\mu^k}(j)\big), \quad i = 1, \dots, n$$

- Optimistic and multistep lookahead versions.

Parametric Approximation and Actor-Critic Schemes


Initial Policy
Evaluate Approximate Cost $\tilde J_\mu(i,\bar r)$ of Current Policy μ
Approximate Policy Evaluation Critic
Generate "Improved" Policy $\bar\mu$ by Lookahead Min Based on $\tilde J_\mu(i,\bar r)$
Policy Improvement Actor

Introduce a differentiable parametric architecture $\tilde J_\mu(i,r)$ for policy evaluation

- Example architectures: A linear featured-based or a neural net.
- Example of approximate policy evaluation: Generate state-cost pairs (i^s,β^s), where β^s is a sample cost corresponding to i^s. Use least squares/regression:

$$\bar r \in \arg\min_r \sum_{s=1}^{q} \left(\tilde J_\mu(i^s,r)-\beta^s\right)^2$$

- β^s is generated by simulating an N-step trajectory starting at i^s, using μ, and adding a terminal cost approximation $\alpha^N \hat J(i_N)$.
- Alternative approximate policy evaluation methods: TD(λ), LSTD(λ), LSPE(λ)

Training, Exploration, and Other Issues

- The training problem
$$\bar{r} \in \arg\min_{r} \sum_{s=1}^{q} \left(\tilde{J}_\mu(i^s, r) - \beta^s \right)^2$$
is well-suited for incremental gradient:
$$r^{k+1} = r^k - \gamma^k \nabla \tilde{J}_\mu(i^{s_k}, r^k)\big(\tilde{J}_\mu(i^{s_k}, r^k) - \beta^{s_k}\big)$$
where (i^{s_k}, β^{s_k}) is the state-cost sample pair that is used at the kth iteration.
- Trajectory reuse: Given a long trajectory $(i_0, i_1, \ldots, i_N)$, we can obtain cost samples for all the states $i_0, i_1, i_2, \ldots$, by using the tail portions of the trajectory.
- Exploration: When evaluating μ with trajectory reuse, we generate many cost samples that start from states frequently visited by μ. Then the cost of underrepresented states may be estimated inaccurately, causing potentially serious errors in the calculation of the improved policy $\overline{\mu}$.
- Bias-variance tradeoff: As the trajectory length N increases, the cost samples β^s become more accurate but also more “noisy."
- Error bounds quantify qualitative behavior; e.g., convergence to within an “error zone."

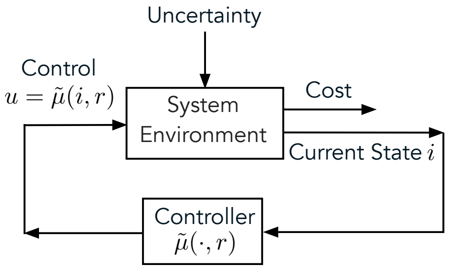
General Framework for Approximation in Policy Space

- Parametrize stationary policies with a parameter vector r; denote them by $\tilde{\mu}(r)$, with components $\tilde{\mu}(i, r)$, $i = 1, \ldots, n$. Each r defines a policy.
- The parametrization may be problem-specific, or feature-based, or may involve a neural network.
- The idea is to optimize some measure of performance with respect to r.

Five contexts where approximation in policy space is either essential or is helpful

- Problems with natural policy parametrizations (like supply chain problems)
- Problems with natural value parametrizations, where a good policy training method works well (like the tetris problem).
- Approximation in policy space on top of approximation in value space.
- Learning from a software or human expert.
- Unconventional information structures, e.g., multiagent systems with local information (not shared with other agents) - Conventional DP breaks down.

Approximation in Policy Space by Optimization-Based Training



Training by Cost Optimization

- Each r defines a stationary policy $\tilde{\mu}(r)$, with components $\tilde{\mu}(i, r)$, $i = 1, \dots, n$.
- Determine r through the minimization

$$\min_r J_{\tilde{\mu}(r)}(i_0)$$

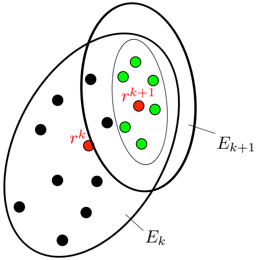
where $J_{\tilde{\mu}(r)}(i_0)$ is the cost of the policy $\tilde{\mu}(r)$ starting from initial state i_0.

- More generally, determine r through the minimization

$$\min_r E\{J_{\tilde{\mu}(r)}(i_0)\}$$

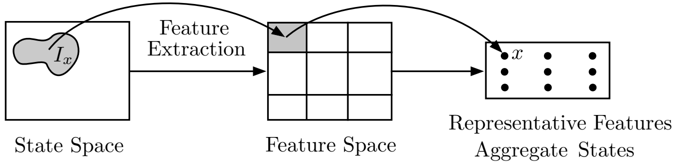
where the $E\{\cdot\}$ is with respect to a suitable probability distribution of i_0.

Training by Random Search - Cross-Entropy Method

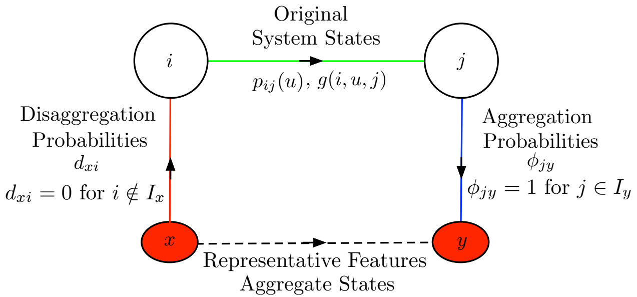


- At the current iterate r^k, construct an ellipsoid E_k centered at r^k.
- Generate a number of random samples within E_k. “Accept" a subset of the samples that have “low" cost.
- Let r^{k+1} be the sample “mean" of the accepted samples.
- Construct a sample “covariance" matrix of the accepted samples, form the new ellipsoid E_{k+1} using this matrix, and continue.
- Limited convergence rate guarantees. Success depends on domain-specific insight and the skilled use of implementation heuristics.
- Simple and well-suited for parallel computation.
- Resembles a “gradient method". Naturally model-free.

Feature-Based Aggregation Framework



Representative feature formation



Transition diagram for the aggregate problem

Some of the Major Points Relating to Aggregation

- It aims to approximate J^*, not J_μ of some policy μ.
- It can yield an arbitrarily close approximation to J^*, with sufficient number of aggregate states.
- Distinction between representative features schemes and their simpler special case, representative states schemes.
- Simulation-based VI and PI methods for solving the aggregate problem.
- Spatio-temporal aggregation: Solve a simpler aggregate problem involving “compression" in space and time.

Concluding Remarks

Some words of caution

- There are challenging implementation issues in all approaches, and no fool-proof methods.
- Problem approximation and hand-crafted feature selection require domain-specific knowledge.
- Training algorithms are not as reliable as you might think by reading the literature.
- Approximate PI involves oscillations and faces challenging exploration issues.
- Recognizing success or failure can be a challenge!
- The RL successes in game contexts are spectacular, but they have benefited from perfectly known and stable models and small number of controls (per state).
- Problems with partial state observation remain a big challenge.

On the positive side

- Massive computational power together with distributed computation are a source of hope.
- Silver lining: We can begin to address practical problems of unimaginable difficulty!
- There is an exciting journey ahead!

Some Words of Relevance

Some old quotes ...

- The book of the universe is written in the language of mathematics. Gallileo
- Learning without thought is labor lost; thought without learning is perilous. Confucius
  (In the language of Confucius' day: learning $\approx$ obtaining knowledge; thought $\approx$ ideas on how to do things)
- Many arts have been discovered through practice, empirically; for experience makes our life proceed deliberately, but inexperience unpredictably. Plato
- White cat or black cat it is a good cat if it catches mice. Deng Xiaoping

... and some more recent ones

- Machine learning is the new electricity. Andrew Ng
  (Electricity changed how the world operated. It upended transportation, manufacturing, agriculture and health care. AI is poised to have a similar impact.)
- Machine learning is the new alchemy. Ali Rahimi and Ben Recht
  (We do not know why some algorithms work and others don't, nor do we have rigorous criteria for choosing one architecture over another ...)

Thank you and good luck!