An Introduction to Stochastic Control and Reinforcement Learning

Subhrakanti Dey and Simone Garatti

Signals and Systems, Dept of Electrical Engineering, Uppsala University
Politechnico Di Milano, Milan

SIDRA Summer School, Bertinoro, Italy

July 2025

Approximate Dynamic Programming and applications

- **Acknowledgements**: Some slides, images and content taken from
  1. Bertsekas and Tsitsiklis, *Neuro-Dynamic Programming*, 1996
  2. Warren Powell, *Approximate Dynamiic Programming* , 2007
  3. Lecture notes and slides by Prof D. P. Bertsekas, available at https://www.mit.edu/ dimitrib/dpbook.html
- Lecture contents: Introduction to Approximate DP due to curse of dimensionality, Approximation architectures, Simulation based methods, Approximate Value and Policy Iteration

A Recap of Value Iteration and Policy Iteration methods

- Finite state and action space MDP
- Bellman equation for dynamic programming: $V^* = TV^*,\ V_\mu = T_\mu V_\mu$

$$V^*(i) = \min_{u \in U(i)} \sum_{j=1}^{N} p_{ij}(u)(g(i,u,j) + \alpha V^*(j)),\ \forall i$$

$$V_\mu(i) = \sum_{j=1}^{N} p_{ij}(\mu(i))(g(i,\mu(i),j) + \alpha V_\mu(j))$$

- Optimality condition: μ is an optimal policy if $T_\mu V^* = TV^*$, or

$$\mu(i) \in \arg\min_{u \in U(i)} \sum_{j=1}^{N} p_{ij}(u)(g(i,u,j) + \alpha V^*(j))$$

A Recap of Value Iteration and Policy Iteration methods

- Value Iteration: For any $V \in \mathbb{R}^n$

$$V^*(i) = \lim_{k \to \infty} (T^k V)(i), \forall i$$

- Policy Iteration: Given policy μ^k, find V_{μ^k} (policy evaluation) by solving

$$V_{\mu^k}(i) = \sum_{j=1}^{N} p_{ij}(\mu^k(i))(g(i, \mu^k(i), j) + \alpha V_{\mu^k}(j))$$

- Then do policy improvement by finding

$$\mu^{k+1}(i) \in \arg \min_{u \in U(i)} \sum_{j=1}^{N} p_{ij}(u)(g(i, u, j) + \alpha V_{\mu^k}(j))$$

- For large state and action spaces policy evaluation (requires solving $N \times N$ equations) is computationally burdensome, similarly for solving for value function at each iteration can involve exponential complexity in state space, worse for continuous state space, as the value function has no closed form solution (curse of dimensionality)

Approximate Dynamic Programming (ADP)

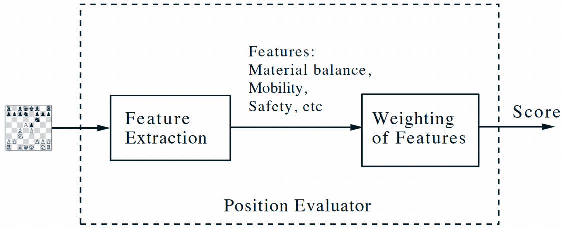
- ADP allows the application of DP towards very large or infinite state space
- A fertile research area since the 1980's
- We will only consider the finite state action space discounted MDP scenario
- We will consider Approximaton in Value-space and Approximation in Policy Space, which are simulation based methods
- We won't discuss other methods such as *state aggregation* or *rollout* based approaches.
- simulations are useful when exact models are not available but one can generate samples by computer simulation, allowing averaging over samples to compute expectations

Approximation in Value Space

- Approximate V^* or V_μ from a parametric class $\tilde{V}(i; r)$ where i is the current state and r is a set of tunable parameters, and use $\tilde{V}$ instead of V^* in computations
- Questions: How to choose the parametric forms and how to tune the weights?
- Problem insight may help, and a simulator may be used when there is no mathematical model of the system
- One can also use parametric approximations for the Q functions (state action value function)
- Approximation architectures can be linear (simple) or nonlinear (e.g. a neural network), which can give a richer approximation (implying linear or nonlinear dependence on the parameter vector r)

Approximation in Value Space

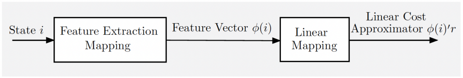
- In the computer chess example, one can think of the board position as states and moves as actions
- Use a *feature* based position evaluator that assigns a score (like a Q value) to each position/move



- Relatively few special features and weights can do the job

Approximation in Value Space

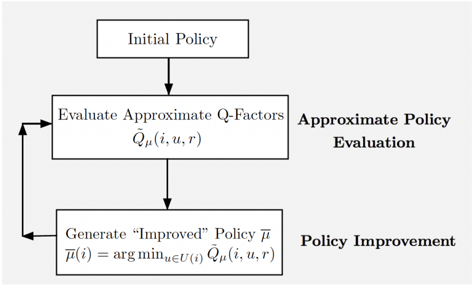
- With few well chosen features (which often encode much of the inherent nonlinearity in the cost function), one can use just a linear architecture $\tilde{V}(r) = \mathbf{\Phi r} = \sum_j \Phi_j r_j$.


State i
Feature Extraction Mapping
Feature Vector $\phi(i)$
Linear Mapping
Linear Cost Approximator $\phi(i)'r$

- The feature space (columns Φ_j can be chosen as polynomial or radial basis functions etc.)
- One can also choose a special set of states and use a parameter vector with an element for each of these states. For the chosen states, use $\tilde{V}(i;r) = r_i$ and for those states which are not in the chosen set, use interpolation of some kind.
- In the game of Tetris with more than 2^{200} number of states, only 22 features are enough, recognized by the players as capturing important aspects of the board position

Approximating policy iterations

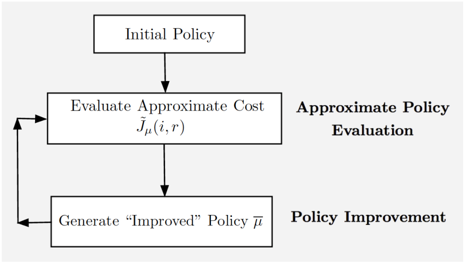
- Use simulation to estimate the value function V_μ or the Q-factor Q_μ for a particular initial policy μ
- Then generate improved policy $\bar{\mu}$ by minimizing the approximate V or Q.


Initial Policy
Evaluate Approximate Q-Factors $\tilde{Q}_\mu(i,u,r)$
Approximate Policy Evaluation
Generate "Improved" Policy $\overline{\mu}$
$\overline{\mu}(i) = \arg\min_{u\in U(i)} \tilde{Q}_\mu(i,u,r)$
Policy Improvement

- One can similarly approximate the optimal Q-factors or the optimal cost function V^* by iteratively simulating the Q-factors from the value functions, and then finding the optimal cost, or by finding the optimal parameter vector to minimize the Bellman error metric

Approximation in Policy Space

- Start with initial policy μ, and then evaluate it using a linear parametric approximation of the cost function $\tilde{J}_\mu(\mathbf{r}) = \mathbf{\Phi r}$, where $\mathbf{\Phi}$ is a full rank $n \times s$ matrix of the basis functions, and the i-th row denoted by $\Phi^T(i)$.


Initial Policy
Evaluate Approximate Cost $\tilde{J}_\mu(i,r)$
Approximate Policy Evaluation
Generate "Improved" Policy $\overline{\mu}$
Policy Improvement

- Policy improvement

$$\bar{\mu}(i) = \arg \min_{u \in U(i)} \sum_{j=1}^{N} p_{ij}(u)(g(i,u,j) + \alpha \Phi^T(j)\mathbf{r})$$

Approximation in Policy Space

- Error Bound: If $\max_i |\tilde{V}_{\mu^k}(i, \mathbf{r}) - V_{\mu^k}(i)| \leq \delta$, then

$$\lim_{k \to \infty} \max_i (V_{\mu^k}(i) - V^*(i)) \leq \frac{2\alpha\delta}{(1-\alpha)^2}$$

- If policy improvement is also approximate

$$\max_i |(T_{\mu^{k+1}}\tilde{V}(i,r) - (T\tilde{V})(i,r)| \leq \epsilon$$

Then

$$\lim_{k \to \infty} \max_i (V_{\mu^k}(i) - V^*(i)) \leq \frac{\epsilon + 2\alpha\delta}{(1-\alpha)^2}$$

Approximation in Policy Space: Practical Issues

- The method makes steady progress upto a certain point and then the iterates oscillate (unpredictably) within a neighbourhood of V^*.
- Since it starts with a specific initial policy μ_0, it may not visit all the states and the cost-to-go approximation of the underrepresented states can be quite bad
- Inadequate exploration: This also affects the policy improvement outcome.
- Possible remedies: Retart simulations frequently from a rich set of initial states, occasionally use a randomly selected control other than the policy, use two different independent Markov chains - one to generate the transition sequence, and another to generate the state sequence.

Approximate Policy Evaluation

- Direct Policy Evaluation: Cost samples generated by simulations and optimization via Least Squares (to find the optimal parameters)
- Indirect Policy Evaluation: Solve a projected equation $\mathbf{\Phi r} = \Pi T_\mu(\mathbf{\Phi r})$ where Π is a projection with respect to a suitable Eucludiean norm
- To calculate Π, one can use simulation based methods such as TD(λ), LSTD, LSPE etc.. (we learn more about these tomorrow from Simone)