Dynamic Programming Lecture #16

Outline:

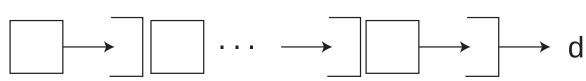
- Approximate Dynamic Programming
- Temporal Difference Learning

Approximate DP & the Curses

- Curse of dimensionality:
  - Large state-space
  - Large control-space
- Example: Inventory control with failure-prone machines

$$x^+ = x + \alpha u - d, \quad \begin{cases} x & \text{inventory} \\ u & \text{production} \\ d & \text{demand} \\ \alpha & \text{machine state} \in \{0,1\} \end{cases}$$

- Probabilities: $p_{01}(u)$ & $p_{10}(u) = ???$
- $\#$ states $= 2\times$ $\#$ allowable parts per machine
- Now consider N-machines:



- $\#$ states $= 2^N \times$ ($\#$ allowable parts per machine)^N
  - 3 parts/machine & 8 machines $\rightarrow 1,700,000$ states
  - 3 parts/machine & 9 machines $\rightarrow 10,000,000$ states
- Curse of modeling: Need knowledge of $p_{ij}(u)$

Facing the Curses

- Approaches:
  - Use experience based updating (online algorithms)
  - Use complexity reduction approximations (cost function parametrization)
  - Use simulation based planning (receding horizon policy evaluation)
- Central point: Approximation of (optimal) cost-to-go
- Suppose $\hat{J}(\cdot)$ is an approximate cost-to-go, and consider

$$\mu(i) = \arg\min_{u \in U(i)} g(i,u) + \sum_j p_{ij}(u)\hat{J}(j)$$

  - If $\hat{J}$ approximates J^*, then above policy should be near optimal.
  - If $\hat{J}$ approximates J_μ, then above policy represents policy update step of policy iteration.

Value Function Approximation

- Impose a structured form of J:

$$J(i) = \Phi(i; r)$$

- Particularly convenient form: Feature vectors.

$$J(i) = \sum_{\ell=1}^{L} \phi_\ell(i) r_i = \phi^{\mathrm{T}}(i) r$$

- Example: Tetris features
  - Column heights
  - Column height differences
  - Maximum column height
  - Number of "holes"
- Basis coefficients r_i determine relative importance of features.
- Approximation based policy:

$$\mu(i; r) = \arg\max_{u \in U(i)} g(i, u) + \sum_{j=1}^{n} p_{ij}(u) \Phi(j; r)$$

Temporal Difference Learning

- Initial focus: Autonomous Systems

$$x_{t+1} = f(x_t, w_t)$$

Note there is no notion of control:

- Consider approximating a value function of the form:

$$J^*(x) = E\left[\sum_{t=0}^{\infty} \alpha^t g(x_t) \Big| x_0 = x\right]$$

with a function of the form

$$\tilde{J}(x, r) = \sum_{k=1}^{K} r(k)\phi_k(x)$$

- General idea: Refine weights through simulated play and observation of results
  - Let r_t be the weights at time t
  - Initial estimate of cost to go from x_t: $\phi(x_t)r_t^T$
  - Improved estimate of cost to go from x_t: $g(x_t) + \alpha \cdot \phi(x_{t+1})r_t^T$
- Define temporal difference (improved estimate - original estimate)

$$d_t = \left(g(x_t) + \alpha \cdot \phi(x_{t+1})r_t^T\right) - \left(\phi(x_t)r_t^T\right)$$

- Goal: Use temporal difference to adjust weights

Temporal Difference Learning

- Temporal Difference Learning: Protocol for adjusting weights

$$r_{t+1} = r_t + \gamma_t \cdot d_t \cdot z_t$$

where $\gamma_t \in [0, 1]$ is step-size and $z_t \in R^K$ is a direction vector and of the form:

$$z_t = \sum_{\tau=0}^{t} (\alpha\lambda)^{t-\tau} \phi(x_\tau)$$

where $\lambda \in [0, 1]$ is a tuning parameter that scales past basis vectors.

- Referred to a temporal different learning with λ, i.e., $TD(\lambda)$
- Special case: $\lambda = 0$, i.e., $TD(0)$, which results in update of form

$$r_{t+1} = r_t + \gamma_t \cdot d_t \cdot \phi(x_t)$$

- Fact: There exists an $r^{(\lambda)}$ such that:
  - $\lim_{t\to\infty} r_t \to r^{(\lambda)}$ with probability 1
  - $\tilde{J}\left(\cdot, r^{(\lambda)}\right)$ performs close to best approximation function using basis functions, i.e.,

$$||J^* - \tilde{J}(\cdot, r^{(\lambda)})|| \leq \rho||\Pi J^* - J^*||$$

  for some $\rho > 1$ and some norm. ΠJ^* is best approximation function.

- Key technical assumptions:
  - Step size $\gamma_t \to 0$ at "right" rate (common choice $\gamma_t = 1/t$)
    * $\sum_{t=0}^{\infty} \gamma_t = \infty$
    * $\sum_{t=0}^{\infty} (\gamma_t)^2 < \infty$
  - All states visited infinitely often.

Gibbs distribution

- New Focus: Controlled system with stationary policy μ

$$x_{t+1} = f(x_t, \mu(x_t), w_t)$$

- Temporal difference learning can be used to approximate J_μ. How do we ensure all states are visited infinitely often?
- Answer: Add noise to the policy μ
- Probability simplex: (assume $U = U(x)$ for all x)

$$\Delta = \left\{ p \in \mathbb{R}^{|U|} : p_i \geq 0 \ \& \ \sum_j p_j = 1 \right\}$$

(suppress $|U|$ in notation)

- Compare:

$$u \sim \textbf{rand}[p] = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad vs. \qquad u \sim \textbf{rand}[p'] = \begin{bmatrix} \epsilon \\ \epsilon \\ 1 - 5\epsilon \\ \epsilon \\ \epsilon \\ \epsilon \end{bmatrix}$$

- Note: p' is approximately p but induces more exploration
- Utility: Ensure that every state visited infinitely often
- Question: Heterogeneity in noise? i.e., better states more likely to be visited?

Gibbs distribution, cont

- Suppose

$$\mu(x) = \arg\max_{u \in U} \; G(x,u)$$

- Now define Gibbs distribution or "soft-max":

$$\sigma_{smax}(x;T) \in \Delta$$

by

$$\sigma_{smax}(x;T) = \frac{1}{Z}\begin{pmatrix} e^{G(x,u_1)/T} \\ e^{G(x,u_2)/T} \\ \vdots \\ e^{G(x,u_{|U|})/T} \end{pmatrix}$$

where Z is a normalizing factor to assure $\sigma_{smax}(x;T)$ is on the simplex, i.e.,

$$Z = e^{G(x,u_1)/T} + e^{G(x,u_2)/T} + \ldots + e^{G(x,u_{|U|})/T}$$

- Details: "Temperature" $T > 0$
- Main idea:
  - For high temperatures ($T \gg 1$), approximates uniform distribution
  - For low temperatures ($T \ll 1$), approximates $\sigma_{max}(x)$
- Example: Let $G(x,u_1) = 1$ and $G(x,u_2) = 2$

$$\frac{1}{Z}\begin{pmatrix} e^{1/0.1} \\ e^{2/0.1} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad vs \quad \frac{1}{Z}\begin{pmatrix} e^{1/1} \\ e^{2/1} \end{pmatrix} = \begin{pmatrix} 0.27 \\ 0.73 \end{pmatrix} \quad vs \quad \frac{1}{Z}\begin{pmatrix} e^{1/5} \\ e^{2/5} \end{pmatrix} = \begin{pmatrix} 0.45 \\ 0.55 \end{pmatrix}$$

- Works for utility maximization. Must put in '-' sign for cost minimization

Controlled Temporal Difference Learning

- Goal: Find stationary policy μ that optimizes

$$J^*(x) = E\left[\sum_{t=0}^{\infty} \alpha^t g(x_t, \mu(x_t)) \Big| x_0 = x\right]$$

  where $\alpha \in [0, 1]$

- Algorithmic thoughts:

  1. Fix structure of approximation functions

  $$\tilde{J}(x, r) = \sum_{k=1}^{K} r(k)\phi_k(x)$$

  2. Fix stationary policy μ^k and simulate with softmax
  3. Use temporal difference learning to approximate J_{μ^k}

  $$r_0^k \rightarrow r_1^k \rightarrow r_2^k \rightarrow \ldots \rightarrow r^{(\lambda,k)}$$

  4. Perform policy improvement: $\mu^k \rightarrow \mu^{k+1}$
  5. Use temporal difference learning to approximate $J_{\mu^{k+1}}$ and simulate with softmax

  $$r_0^{k+1} = r^{\lambda,k} \rightarrow r_1^{k+1} \rightarrow r_2^{k+1} \rightarrow \ldots \rightarrow r^{(\lambda,k+1)}$$

  6. Repeat

- Note: Time-scale separation, i.e., evaluation then improvement

- Is time-scale separation necessary? In practice, it does not appear so.

Controlled Temporal Difference Learning

- Recap: Tetris objective

$$\max \lim_{N\to\infty} E\left\{\sum_{k=0}^{N} g(x_k, u_k)\right\}$$

- Assume a linear basis approximation for the value function:

$$\phi(i)r^T \approx \lim_{N\to\infty} E\left\{\sum_{k=0}^{N} g(x_k, u_k)|x_o = i\right\}$$

- Algorithmic procedure
  - Step 1: Simulate policy using approximate cost to go with current weights
    - State/weights at time t: x_t, r_t
    - Action at time t: $u_t \sim \sigma_{smax}(x;T)$ where $T > 0$ and for any $u_t \in U(x_t)$

$$G(x_t, u_t) = E_{w_t}\Big[g(x_t, u_t) + \phi(f(x_t, u_t, w_t))r_t^T\Big]$$

    - State at time $t+1$: $x_{t+1} \sim f(x_t, u_t, w_t)$
  - Step 2: Evaluate temporal difference

$$d_t = g(x_t, u_t) + \phi(x_{t+1})r_t^T - \phi(x_t)r_t^T$$

  - Step 3: Revise weights

$$r_{t+1} = r_t + \gamma_t \cdot d_t \cdot z_t$$

    In the case of $TD(0)$ we have

$$r_{t+1} = r_t + \gamma_t \cdot d_t \cdot \phi(x_t)$$

  - Repeat
- Tends to work well in practice. No theoretical guarantees.
- For more information see: Neuro-Dynamic Programming: Overview and Recent Trends by Benjamin Van Roy