Elements of Reinforcement Learning

- **Policy: way learning algorithm behaves (mapping from state to action)**
- **Reward function: Mapping of state action pair to reward or cost**
- **Value function: long term reward, total weighted or unweighted reward in present and future**
- **Model: mimic behavior of environment**

Evaluative Feedback Example

- Consider n-armed bandit problem: at every instant must choose one of n actions with the goal of maximizing rewards.
- Expected reward for action *a*, $Q^*(a)$ and estimated value of *t*th play $Q_t(a)$. Set

$$Q_t(a) = \sum r_i / k_a$$

where k_a is number of times action *a* taken

- Exploration versus exploitation
- How to choose action *a*

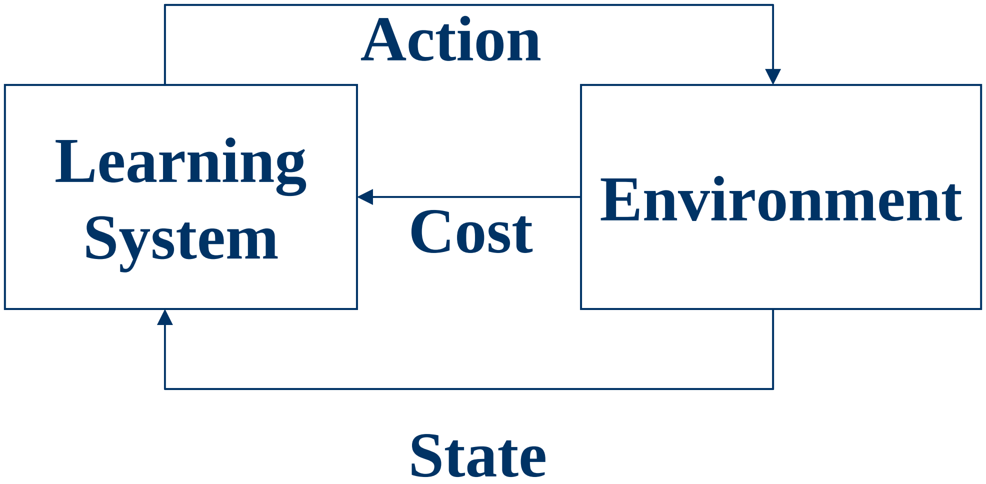
Policies for n-armed bandit problem

- **Greedy policy: $Q_t(a^*) = \max_a Q_t(a)$. No exploration, initially does well, but poor long term performance.**
- **ε-greedy policy: same as greedy policy, but with prob. ε randomly choose policy. Some exploration, does better than greedy policy.**
- **Softmax Action selection: weight actions probabilistically with temperature parameter (Gibbs distribution).**
- **Reinforcement methods: keep track of payoffs (reinforcement as opposed to action-value method).**
- **Pursuit methods: use both action-value estimates and action preferences.**

Summary of example

- Exploitation versus Exploration
  - Algorithms presented
  - What is the proper balance?
- Learning schemes
  - Supervised learning: instructed what to do
  - Evaluative learning: try different actions and observe rewards (allows more control of environment)
- Non associative learning: trial and error learning not associated with situation or state of the problem (only one state)

Reinforcement Learning Model



- Exploration versus exploitation
- Learning can be slow

Finite Markov Decision Processes

- Parameters
  - State: $X(n) = x(n)$, N states
  - Action: $A(n) = a_{ik}$ (action from state i performing action k)
  - Transition probability: $p_{ij}(a) = P(X(n+1) = j \mid X(n) = i, A(n) = a)$
  - Cost: $r(i,a,j)$ and discount factor γ, with $0 \leq \gamma < 1$
  - Policy: $\pi = \{u(0), u(1), \ldots\}$, policy mapping states into actions (stationary and nonstationary)

Value Functions

- Cost or value function (infinite horizon, discounted)

$$J^{\pi}(i) = E\,(\Sigma\ \gamma^{n}\ r(x(n),u(x(n)),x(n+1))|x(0)=i)$$

averaged over Markov chain x(1),x(2), …

- Action-value function

$$Q^{\pi}(i,a) = E\,(\Sigma\ \gamma^{n}\ r(x(n),u(x(n)),x(n+1))|x(0)=i,a(0)=a)$$

averaged over Markov chain x(1),x(2), …

Find policy π that minimizes $J^{\pi}(i)$ for all initial states i

Recursive expression for value function

$$J^{\pi}(i) = E\left(\Sigma\, \gamma^{n}\, r(x(n),u(x(n)),x(n+1))|x(0)=i\right)$$

$$=E(r(i,u(i),j) + \gamma\Sigma\, \gamma^{n}\, r(x(n+1),u(x(n+1)),x(n+2))|x(1)=j)$$

$$= \Sigma_{a}\, \Sigma_{j}\, \pi(i,a)\, p_{ij}\,(a)\, (r(i,a,j) + \gamma\, J^{\pi}(j)\,)$$

Bellman equation for J^{π} allows for calculation of value function for policy π.

Equation can be solved iteratively or directly.

Optimal Value Function

Want to find optimal policy to maximize value function

$$J^*(i) = \max_{\pi} J^{\pi}(i)$$

Can express optimal value function in terms of action-value function as

$$J^*(i) = \max_{u(i)} Q^*(i,u(i))$$

where $Q^*(i,u(i)) = \max_{\pi} Q^{\pi}(i,u(i))$

Then can find a recursive expression for $J^*(i)$ by expanding RHS of equation similar to method found in previous slide for value function.

MDP Solution and Bellman Equation

- Use dynamic programming, can formulate cost function in terms of Bellman's Optimality equation

$$J^*(i) = \max_u E_{x(i)} [r(i,u(i),x(i)) + \gamma J^*(x(i))]$$

- Current cost: $c(i,u(j)) = E_{x(i)} [r(i,u(i),j)] = \Sigma_{j=1,N} p_{ij} \; r(i,u(i),j)$
- Rewrite Bellman's equation

$$J^*(i) = \max_u [c(i,u(i)) + \gamma \Sigma_{j=1,N} p_{ij} J^*(j)]$$

- System of N equations with (equation/ state) and minimization

Policy Evaluation and Improvement

- Policy Evalution:For a given policy we can iteratively compute value function

$$J_{k+1}^{\pi}(i) = \Sigma_a \Sigma_j \pi(i,a)\, p_{ij}(a)\, (r(i,a,j) + \gamma J_k^{\pi}(j))$$

Iterative algorithm converges.

- Policy Improvement: Q function can be expressed iteratively as

$$Q^{\pi}(i,a) = c(i,a) + \gamma \Sigma_{j=1,N}\, p_{ij}(a)\, J^{\pi}(j)$$

u is said to be greedy with respect to $J^u(i)$ if

$$u(i) = \max_a Q^{\pi}(i,a) \text{ for all } i$$

Policy Iteration

1) Policy evaluation: $J^{u}(i)$

Cost to go function needs recomputation

$$J^{u_n}(i) = c(i,u_n(i)) + \gamma \Sigma_{j=1,N}\, p_{ij}\, (u_n(i))J^{u_n}(j)$$

Solve set of N linear equations directly or iteratively.

2) Policy improvement: $u_{n+1}(i) = \mathrm{argmax}_a\, Q^{u_n}(i,a)$

Value Iteration

- **Initialization: start with initial value $J_0(i)$**
- **Iterate: $Q(i,a) = c(i,a) + \gamma \sum_{j=1,N} p_{ij} J_n(j)$**

  $$J_{n+1}(i) = \max_a Q(i,a)$$
- **Continue until $|J_{n+1}(i) - J_n(i)| < \varepsilon$**
- **Compute policy: $u^* = \arg\max_a Q(i,a)$**

Dynamic Programming Comments

- Number of states often grows exponentially as number of state variables. (Bellman’s curse of dimensionality)
- For large state spaces it is infeasible to search entire state space to perform DP steps. Asynchronous DP used where partial searches and updates are made of state space.
- DP programs run polynomially in number of states and actions.
- GPI (Generalized Policy Iteration) often used instead of PI where Policy Evaluation and Policy Improvement done together.
- DP assumes complete knowledge of environment.

Approximate Dynamic Programming

- Incomplete information (do not know Markov transition probabilities)
- Curse of dimensionality
- Opt for suboptimal policy where J*(i) replaced by approximations of J*(i) that can consist of table lookup or parameterized by set of weights
- Use Monte Carlo simulations to learn policy
- Q learning

Q Learning Algorithm

- **Define Q function**

$$Q^*(i,a) = \Sigma_{j=1,N}\, p_{ij}\,(a)\,(r(i,a,j) + \gamma \max_b Q^*(j,b))$$

$$J^*(i) = \max_a Q^*(i,a)$$

- **Use iterative learning to learn Q function**

$$Q_{n+1}(i,a) = (1- \mu(i,a))\, Q_n(i,a) + \mu(i,a)(r(i,a,j) + \gamma J_n(j))$$

where j is random successor state with

$$J_n(j) = \max_b Q_n(i,b)$$

- **Monte Carlo Simulations: update only applies to current state-action pair all other pairs are not updated**

Q Learning Comments

- Convergence Theorem: Q Learning algorithm converges almost surely to optimal Q function given certain conditions on step size (stochastic approximation conditions) and all state pairs are visited infinitely often
- Representations: Table lookup works well, but networks parameterized by weights often learn very slowly
- Exploration vs. exploitation: ensure all state-action pairs are explored while also minimizing cost to go function

Temporal Difference Learning

- Given a learning sequence where a termination occurs and a reward is given how do we learn?
- Credit assignment to each training input in the sequence can be performed using temporal difference learning
- Iterative learning algorithms can then be established with inputs and target outputs
- Class of TD(λ) algorithms where $0 \leq \lambda \leq 1$
- Learning much slower than supervised learning

Reinforcement Learning Applications

- Backgammon
- Navigation
- Elevator control
- Helicopter control
- Computer network routing
- Sequential detection
- Dynamic channel allocation (cellular system)

References

- **R. Sutton, A. Barto, *Reinforcement Learning An Introduction*, MIT Press, Cambridge, MA, 1998.**
- **D. Bertsekas, J. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA, 1996.**
- ***Handbook of Learning and Approximate Dynamic Programming*, Editors J. Si, A. Barto, W. Powell, D. Wunsch, Wiley-IEEE Press, 2004.**