Fundamentals of Reinforcement Learning

Dr. Fayazur Rahaman Mohammad,
Dept. of ECE, Mahatma Gandhi Institute of Technology, Hyderabad.

- **Fundamentals**
  - **Action Values**
    - **Expected Reward** → $q_*(a) \triangleq \mathbb{E}[R_t|A_t=a] = \sum_r p(r|a)r$ → Goal is to maximize: $\underset{a}{\text{argmax}}\ q_*(a)$
    - **Sample Average**
      - $Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i}{t-1}$ → Greedy action: $a_g \triangleq \underset{a}{\text{argmax}}\ Q(a)$
      - Incremental update: $Q_{n+1} = Q_n + \frac{1}{n}(R_n - Q_n)$ → Decaying past rewards: $= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i$
  - **Tradeoff**
    - **Exploration** → Improve knowledge for long-term
    - **Exploitation** → Exploit knowledge for short-term
    - **Action Selection**
      - ϵ-greedy action selection → $A_t \leftarrow \begin{cases} \underset{a}{\text{argmax}}\ Q_t(a) \text{ with } 1-\epsilon \\ a \sim \mathcal{U}\{a_k\} \text{ with } \epsilon \end{cases}$
      - Optimistic initial value → $Q_{n+1} \leftarrow Q_n + \alpha(R_n - Q_n)$ → drives only early exploration, not suited for non-stationary
      - Upper-confidence bound → $A_t \triangleq \underset{a}{\text{argmax}}\ \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}\right]$ → Exploitation: $Q_t(a)$, Exploration: $c\sqrt{\frac{\ln t}{N_t(a)}}$
  - **Markov (MDP)**
    - **General Framework**
      - Sequential decision-making
      - Action influences future situation → $p(s', r|s, a)$ → $\sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s', r|s, a) = 1$
    - **Agent** → Goal of agent is to maximize expected return
      - Episodic tasks → $\mathbb{E}[G_t] \triangleq \mathbb{E}[R_{t+1} + \cdots + R_T]$
      - Continuing tasks → $\mathbb{E}[G_t] \triangleq \mathbb{E}[R_{t+1} + \cdots] = \infty?$ → Discount future rewards: $G_t \triangleq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_t + \gamma G_{t+1}$, $\gamma = 0 \implies$ short-sighted and $\gamma = 1 \implies$ far-sighted
    - **Given State S**
      - Policy function distribution over actions
        - Deterministic policy $\pi(s) = a$ → possible count $|\mathcal{A}|^{|\mathcal{S}|}$
        - Stochastic policy $\pi(a|s)$ → $\sum_{a\in\mathcal{A}(s)} \pi(a|s) = 1$
      - State-value = expected return into the future → $v_\pi(s) \triangleq \mathbb{E}_\pi[G_t|S_t=s] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$
      - Action-value = expected return after taking action a → $q_\pi(s, a) \triangleq \mathbb{E}_\pi[G_t|S_t=s, A_t=a]$, $= \sum_{s',r} p(s', r|s, a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')]$ → Can solve directly for small MDPs
    - **Optimality**
      - Optimal policy π_* achieves the highest value possible $\forall s$ one or more optimal policies exist → $v_{\pi_*}(s) \triangleq \mathbb{E}_{\pi_*}[G_t|S_t=s] = \underset{\pi}{\max}\ v_\pi(s)$ → share optimal value v_*
      - Optimal Value functions, $v_*(s)$
        - $v_*(s) = \underset{a}{\max} \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$ → Bellman optimality eq for v_* → solution is a deterministic policy π_*
        - $\implies \pi_*(s) = \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')]$ → Need $p(s', r|s, a)$ and $v_*(s')$ to obtain $\pi_*(s)$
      - Optimal Action-Value functions, $q_*(s, a)$
        - $q_{\pi_*}(s, a) = \underset{\pi}{\max}\ q_\pi(s, a)$ → share optimal action-value $\mathbf{q}_*$
        - $q_*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \underset{a'}{\max}\ q_*(s', a')]$ → Bellman optimality eq for $\mathbf{q}_*$
        - $\implies \pi_*(s) = \underset{a}{\text{argmax}}\ q_*(s', a)$ → Need just $q_*(s', a)$ to obtain $\pi_*(s)$
  - **Dynamic (DP)**
    - **Introduction** → DP uses $p(s', r|s, a)$ and B. eqs to obtain v_*, q_* and π_*
    - **Policy**
      - Evaluation: Task of determining v_π given π → Iterative policy evaluation $v_k \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$ → For any v_0, $\lim_{k\to\infty} v_k = v_\pi$
      - Control: Task of improving a policy → Policy improvement theorem
        - $q_\pi(s, \pi') \geq q_\pi(s, \pi), \forall s \implies \pi' \geq \pi$
        - $\pi'(s) = \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$ → greedified policy
      - Iteration: evaluation $V \rightsquigarrow v_\pi$ and improvement $\pi \rightsquigarrow \text{greed}(V)$ → only at optimal policy $\pi_* \leftrightarrow v_*$
      - Generalized Policy Iteration (GPI): Value iteration → $V(s) \leftarrow \underset{a}{\max} \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
        - single step for eval. and impr.
        - Asynch. method → updating states in random order efficiently propogates value funciton
    - **Conclusion**
      - Advantages
        - Bootstrapping for policy evaluation → use of value estimates of successive states unlike Monte Carlo
        - Policy improvement Theorem → is exponentially faster than Brute-Force search
      - Disadvantages → Pure DP requires env model for computations → tedious to obtain $p(s', r|s, a)$

Suggestions: mfrahaman_ece@mgit.ac.in, fayazur.mohammad@gmail.com

Sample-based Learning Methods

Dr. Fayazur Rahaman Mohammad,
Dept. of ECE, Mahatma Gandhi Institute of Technology, Hyderabad.

- **Samples**
  - **Monte Carlo (MC)**
    - **Estimates values by average**
      - over large number of random samples → Policy Evalution $\mathbb{E}_\pi[G_t|S_t = s]$
        - → MC observes multiple returns from s → averages to obtain expected value, larger the sample count, better the average
        - → returns are observed only at end of episodic tasks
      - don't need env model → learns independently of other states
    - **Prediction for episodic tasks**
      - estimating state-values $V \approx v_\pi$ → $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$ → After each episode iterate backward update $G \leftarrow \gamma G + R_{t+1}$ avg. over multiple G for each S_t
      - estimating action-values given s, a → $q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$ are useful for learning a policy π → switch to better action using $\underset{a}{\text{argmax}}\ q_\pi(s, a)$ → needs $q_\pi(s, a)$ for all actions, needs MC exploration $\forall a$ given s
      - exploration methods
        - → 'Exploring Starts' → Gaurantees episode starts in every state-action pair But, impractical in some cases, Ex: self-driving car → randomly chooses a start-action pair
        - → ϵ-Soft policies
          - → forces agent to continually explore → each action taken with atleast $\frac{\epsilon}{|\mathcal{A}|}$ probability
          - → Ex: ϵ-greedy (stochastic policy) → greedy + occasional random action
      - Generalized Policy Iteration
        - → for estimating $\pi \approx \pi_*$
        - → improvement: $\pi_{k+1}(s) = \underset{a}{\text{argmax}}\ q_{\pi_k}(s, a)$ evaluation: MC prediction → generate episodes, using 'Exploring start' and π → Iterate backward: $G \leftarrow \gamma G + R_{t+1}$, $Q(S_t, A_t) \leftarrow$ avg G, $\pi(S_t) \leftarrow \underset{a}{\text{argmax}}\ Q(S_t, a)$
        - → estimating $\pi \approx \pi_*$ using ϵ-soft policies → generate first episode using arbitrary $\pi \leftarrow \epsilon$-soft, subsequent episodes using updated π → Iterate backward: $A^* \leftarrow \underset{a}{\text{argmax}}\ Q(S_t, a)$, $\pi(a|S_t) \leftarrow \begin{cases} \frac{\epsilon}{|\mathcal{A}(S_t)|} + (1-\epsilon) & \text{if } a = A^* \\ \frac{\epsilon}{|\mathcal{A}(S_t)|} & \text{if } a \neq A^* \end{cases}$
  - **Off-policy**
    - **Introduction**
      - On-policy disadvantage → improves and evaluates the policy being used to select actions → suboptimal policies for both acting and learning
      - Helps deal with exploration problem
      - improve and evaluate a different policy from the one used to select actions
        - → Target policy $\pi(a|s)$ → learn values for π, ex: optimal policy
        - → Behaviour policy $b(a|s)$ → select actions from b, ex: exploratory policy
      - Importance sampling → estimate the expected value $\mathbb{E}_\pi[X]$ using samples from $b(x)$ → $\mathbb{E}_\pi[X] \approx \frac{1}{n}\sum_{i=1}^{n} x_i \rho(x_i)$ where $x_i \sim b$ and $\rho(x) = \frac{\pi(x)}{b(x)}$
    - **MC prediction** → Estimating state-values $V \approx v_\pi$ → $V_\pi(s) = \mathbb{E}_b[\rho_{t:T-1} G_t|S_t = s]$, where $\rho_{t:T-1} = \frac{\mathbb{P}(\text{ trajectory under } \pi)}{\mathbb{P}(\text{ trajectory under } b)}$ → $\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$ → For each episode following b: Iterate backward: $G \leftarrow \gamma W G + R_{t+1}$, $V(S_t) \leftarrow$ avg G, $W \leftarrow W\frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
  - **Temporal Difference (TD)**
    - **TD introduction**
      - → Disadvantage of estimating values from returns → $v_\pi = \mathbb{E}_\pi[G_t|S_t = s]$, $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$ → To compute return G_t, need estimates of full trajectories → Implies no learning during the episode
      - → TD allows incremental learning, before end of episode → Bootstraps as in DP and learns from env as in MC → Do not require a model of the env Converges faster than MC
    - **TD to estimate value function** → using bootstrapping, we have recursive value function $v_\pi(s) = R_{t+1} + \gamma v_\pi(S_{t+1})$ → by replacing $G_t \approx R_{t+1} + \gamma V(S_{t+1})$, we have incremental MC update rule as $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
      - → Tabular TD(0) for estimating v_π → For each episode: Initialize S; For each step: $A \leftarrow$ action from π for S; Observe R, S'; $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$; $S \leftarrow S'$
      - → where TD-error $\delta_t \triangleq [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
    - **GPI with TD: Sarsa** → To find improved policies Policy improvement at every step → Sarsa Algorithm: learns q_π $S_t\ A_t\ R_{t+1}\ S_{t+1}\ A_{t+1}$ → makes prediction about state-action pairs $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$ with $S_{t+1} \sim p(s', r|s, a)$ and $A_{t+1} \sim \epsilon$-greedy over $\pi(a'|s')$ → is an On-policy algo
    - **Q-Learning:** → Directly learns q^* → instead of switching between policy improvement and evaluation → $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t))$ → is off-policy without using importance sampling
    - **Expected Sarsa:** → nearly identical to Sarsa → except that TD-error uses expected estimate of the next action-value instead of a sample of next action-value
      - → $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a') - Q(S_t, A_t))$ → estimates have lower variance, whereas computations are higher
      - → can use off-policy learning: $\sum_{a'} b(a'|S_{t+1}) Q(S_{t+1}, a')$ → Q-learning is a special case when $b = \pi^*$

Suggestions: mfrahaman_ece@mgit.ac.in, fayazur.mohammad@gmail.com

Dr. Fayazur Rahaman Mohammad,
Dept. of ECE, Mahatma Gandhi Institute of Technology, Hyderabad.

- **Dyna**
  - **Intro.**
    - **Objective**
      - TD works on samples whereas DP works on distribution info.
        - Dyna architecture unifies best of both
    - **Model**
      - Models store knowledge of the dynamics
        Given (S_t, A_t), model produces (S_{t+1}, R_{t+1})
        - can be used for Planning, i.e., improve a Policy
          - with simulated experience, agent requires fewer interaction with env. to come-up with a policy
        - Create simulated experience
          - allows Value estimate updates
            - which further allows policy updates
    - **Model Types**
      - Sample model:
        - produces actual outcomes drawn from some underlying probabilities
          - Computationaly inexpensive
            Require less memory
            Can only approx expected outcomes
      - Distribution model:
        - completely specifies the probability of every outcome
          - can compute exact expected outcome
    - **Q-Planning**
      - Planning takes model as input and produces improved policy without having to interact with env
        - recall Q-learning uses experience from env and updates its policy
        - similary Q-planning uses experience from model and updates its policy
          - learing experience occurs at specific time points, planning updates can be used to fill-in the gaps
          - Random-sample one-step tabular Q-planning:
            - Choose (S, A) pair at random from model
              Query the model with (S, A) for (S', R)
              $Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma \max_{a'} Q(S', a') - Q(S, A))$
              Improvement: $\pi(s) = \underset{a}{\text{argmax}}\ Q(s, a)$
            - this can be done in parallel with real experience updates
  - **Dyna Arch.**
    - **Need**
      - Unify direct RL updates and planning updates
    - **Components**
      - Environment
        Policy / value functions
        Direct RL update
        - Agent completes learning episodes using environment experience
        - Environment exp. is used to learn a model
          Model simulates model exp. using search control
          Planning update are performed using model exp.
          - After each learning episode,
            Agent performs N planning updates
            from the states the agent has already visited
            - As a result policy is improved more quickly
              Uses the limited env exp. more efficiently
              However, needs more computations per step
          - Search control is the process of choosing the state from which to simulate the model experience
  - **Tabular Dyna-Q & Dyna-Q+**
    - **Setup**
      - Assumes deterministic model
        i.e., if transition has occured once,
        it will always unfold the same way in future
    - **Algo**
      - Initialize $Q(s, a)$ and $M(s, a)\ \forall s, a$
        Loop forever:
        - $S \leftarrow$ non-terminal state
        - $A \leftarrow \epsilon$-greedy(S, Q)
        - Update $Q(S, A)$
        - $M(S, A) \leftarrow R, S'$
        - Loop n times:
          - $S \leftarrow$ rand observed state
          - $A \leftarrow$ rand action taken in S
          - $R, S' \leftarrow M(S, A)$
          - Update $Q(S, A)$
        - Dyna-Q mixes planning, learning and acting through the value function
        - Models can be inaccurate if they are incomplete or the environment changes
          - Planning with inaccurate model improves the policy w.r.t to model, not the environment
          - Agent need to explore and exploit for the model to stay accurate
            Bonus rewards for exploration
            - Dyan-Q+ Algorithm uses:
              New reward $\leftarrow r + \kappa\sqrt{\tau}$
              where τ is time steps since transition was last tried

Suggestions: mfrahaman_ece@mgit.ac.in, fayazur.mohammad@gmail.com

Dr. Fayazur Rahaman Mohammad,
Dept. of ECE, Mahatma Gandhi Institute of Technology, Hyderabad.

Function Approx.

- **Value function**
  - **Parameterized functions**
    - Tabular methods store learned values for each possible state
      - will become intractably large for most practical scenarios
    - Parameterized function $\widehat{\nu}(s, \mathbf{w})$, takes a state and produces an approximate value
      - Weights $\mathbf{w}$ parameterize the value function. As a result, donot have to store whole table values, instead have to store only weights $\mathbf{w}$
        - this representation is useful for large state spaces
        - Learning algorithms will modify $\mathbf{w}$, instead of the estimated values, Modifying a weight now changes the values of all the states
      - Linear value function approximation
        - value of each state s is represented as a linear function of weights $\widehat{\nu}(s, \mathbf{w}) \triangleq \sum w_i x_i(s) =< \mathbf{w}, \mathbf{x}(s) >$, where $x_i(s)$ are fixed attributes of states called features
          - Linear value function approx. rely on having good features
      - Neural Network is an example of a non-linear function approximation
        - Tabular value fn. is a special case of linear fns. where feature $\mathbf{x}(s_i) = [0\, 0\, \cdots\, 1\, 0]^T$, is an indicative fn for a particular state
          - feature $x_i = 1$ for $s = s_i$ and 0 otherwise
  - **Generalization**
    - Generalization: Applying knowledge about a specific situation to draw conclusion about wider variety of situations
      - is important for faster learning
    - Discrimination: The ability to make the value of two states different
      - need to discriminate between states to accurately represent the values with availability of information
        - Ex: Tabular methods have good discrimination but no generalization. Having both gen. and disc. is ideal
          - Degree of generalization has a major impact on the performance of the algo
  - **Estimating**
    - frame the policy evaluation task $\approx \nu_\pi(S)$ as a supervised learning problem
      - In MC given S, the target is to estimate G where as in TD, target is to estimate one-step bootstrap return $R + \gamma\widehat{\nu}(S, \mathbf{w})$
        - the target $R + \gamma\widehat{\nu}(S, \mathbf{w})$ continously changes in TD
        - we need SL methods that allow bootstrapping and work in on-line setting
  - **On-policy**
    - Policy evaluation under function approx. $\{S, \widehat{\nu}(s, \mathbf{w})\}$, where $\widehat{\nu}(s, \mathbf{w}) \approx \nu_\pi(s)$ requires us to specify an objective
      - Mean squared value error $\overline{VE} = \sum_s \mu(s)[\nu_\pi(s) - \widehat{\nu}(s, \mathbf{w})]^2$ is one possible objective
        - where $\mu(s)$ is the importance attached to each state the fraction of time spent in s when following π
        - the weights $\mathbf{w}$ need to be adapted to minimize $\overline{VE}$
- **Gradient Descent**
  - **Intro.**
    - Gradient Descent can be used to find stationary points of objectives
      - The gradient $\nabla f \triangleq [\frac{\partial f}{\partial w_1} \cdots \frac{\partial f}{\partial w_d}]^T$ gives the direction of steepest ascent
        - Ex: for linear value function $\widehat{\nu}(s, \mathbf{w}) \triangleq \sum w_i x_i(s)$, $\nabla\widehat{\nu}(s, \mathbf{w}) = \mathbf{x}(s)$
        - Goal is to adjust $\mathbf{w}$ to make objective $\overline{VE}$ small, using GD update rule: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha\nabla\overline{VE}(\mathbf{w}_t)$
          - Even a stochastic algo, will tend to converge at local minimum. In some cases converges at global minimum, however $\widehat{\nu} \neq \nu_\pi$
  - **Gradient Monte**
    - Combine GD to minimize value error and monte carlo algo for value estimation
      - After simplification, we have $\nabla\overline{VE} = -\sum_{s\in\mathcal{S}} \mu(s) 2[\nu_\pi(s) - \widehat{\nu}(s, \mathbf{w})]\nabla\widehat{\nu}(s, \mathbf{w})$
        - Computing $\nabla\overline{VE}$ requires summing $\forall s \in \mathcal{S}$ which is not feasible, and also the knowledge of $\mu(s)$
          - Alternatively, Stochastic Gradient descent rule is used $\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha[\nu_\pi(S_i) - \widehat{\nu}(S_i, \mathbf{w}_i)]\nabla\widehat{\nu}(S_i, \mathbf{w}_i)$ i.e., update weights for single state S_i at a time
        - Using Gradient with Monte Carlo estimate, we have $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[G_t - \widehat{\nu}(S_t, \mathbf{w}_t)]\nabla\widehat{\nu}(s, \mathbf{w}_t)$
          - Loop forever:
            - Generate an episode using π
            - Loop for each step $t = 0, 1, \cdots, T-1$
              - $\mathbf{w} = \mathbf{w} + \alpha[G_t - \widehat{\nu}(S_t, \mathbf{w})]\nabla\widehat{\nu}(S_t, \mathbf{w})$
  - **State Aggregation**
    - Value update to a state leads to updates to all the states in the group
      - this is an example of linear function approx.
      - one feature for each group of states, ex: $\mathbf{x}(s) = [1\, 0]^T, \widehat{\nu}(s, \mathbf{w}) = w_1$ when $s \in \mathcal{S}_1$ and $\mathbf{x}(s) = [0\, 1]^T, \widehat{\nu}(s, \mathbf{w}) = w_2$ when $s \in \mathcal{S}_2$
  - **Semi-Gradient TD**
    - Recall: TD algo uses its own estimate to update value function
    - Recall: $\mathbf{w} = \mathbf{w} + \alpha[U_t - \widehat{\nu}(S_t, \mathbf{w})]\nabla\widehat{\nu}(S_t, \mathbf{w})$ GD updates $\mathbf{w}$ for $\widehat{\nu}(S, \mathbf{w})$ to be $\approx U_t$
      - One-step bootstrap target in TD is: $U_t = R_{t+1} + \gamma\widehat{\nu}(S_{t+1}, \mathbf{w})$
        - $\nabla\overline{VE} = \nabla[U_t - \widehat{\nu}(S_{t+1}, \mathbf{w})]^2 = 2(U_t - \widehat{\nu}(S_{t+1}, \mathbf{w}))(\nabla U_t - \nabla\widehat{\nu}(S_{t+1}, \mathbf{w}))$ Instead, we use semi-gradient: where TD-error $\delta_t \triangleq R + \gamma\widehat{\nu}(S', \mathbf{w}) - \widehat{\nu}(S, \mathbf{w})$
          - Loop for each episode:
            - Initialize S
            - Loop for each step: (till terminal)
              - Take action $A \sim \pi(\cdot|S)$, observe R, S'
              - $\mathbf{w} \leftarrow \mathbf{w} + \alpha\,\delta_t\,\nabla\widehat{\nu}(S, \mathbf{w})$
              - $S \leftarrow S'$
  - **Linear TD**
    - Linear methods are simpler to understand and analyze. With good features, LTD can learn quickly and achieve good prediction accuracy
      - With $\widehat{\nu}(S_t, \mathbf{w}) \triangleq \mathbf{w}^T\mathbf{x}(S_t)$, we have $\nabla\widehat{\nu}(S_t, \mathbf{w}) = \mathbf{x}(S_t)$ and TD update rule as: $\mathbf{w} \leftarrow \mathbf{w} + \alpha\,\delta_t\,\mathbf{x}(S_t)$
        - Tabular TD is a special case of linear TD. Ex: with $\mathbf{x}(S_i) = [0\, 0 \cdots\, 0\, 1\, 0]$ we have $\widehat{\nu}(S_i, \mathbf{w}) = w_i$
        - The expectation of TD update is $\mathbb{E}[\nabla\mathbf{w}_t] = \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_t)$, where $\mathbf{b} \triangleq \mathbb{E}[R_{t+1}\mathbf{x}_t]$ and $\mathbf{A} \triangleq \mathbb{E}[\mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^T]$
          - Weights converge to TD fixed point, i.e, when $\mathbb{E}[\nabla\mathbf{w}_{TD}] = \alpha(\mathbf{b} - \mathbf{A}\mathbf{w}_{TD}) = 0, \implies \mathbf{w}_{TD} = \mathbf{A}^{-1}\mathbf{b}$. Further $\overline{VE}(\mathbf{w}_{TD}) \leq \frac{1}{1-\gamma} \min_{\mathbf{w}} \overline{VE}(\mathbf{w})$

Suggestions: mfrahaman_ece@mgit.ac.in, fayazur.mohammad@gmail.com

Dr. Fayazur Rahaman Mohammad,
Dept. of ECE, Mahatma Gandhi Institute of Technology, Hyderabad.

Prediction and Control using Function Approximation-II

Func. Approx.

Features

Linear Value Functions

The features used to construct value estimates are important factors of func. approx

Recall $\nu_\pi(s) \approx \widehat{\nu}(s, \mathbf{w}) = \mathbf{w}^T\mathbf{x}(s)$, each state s is represented by a feature vector $\mathbf{x}(s)$. In case of tabular method $\mathbf{x}(s)$ is a binary vector (one-hot coding)

- One-hot encoding is not feasible due to its memory requirement when state-space is large
- Ex: In a two-dimensional continuous state space, state aggregation is used to represent near-by states
  - By allowing overlaps in state aggregation we gain more flexibility, and is called coarse coding
  - Ex: $\mathbf{x}(s) = [0\,1\,0 \cdots 0\,1\,0]$ represents a state aggregated into two over-lapping groups

Coarse Coding

Updates to the value of one state changes the values of all states within receptor fields of the active features

- Larger receptor fields will lead to larger gen., thereby increasing the speed of learning
  - The direction of generalization depends on the shape of the field
    - Ex: For elliptic fields, gen. is more in the direction of the major-axis
- Accuracy of the estimates depends on the state discrimination
  - Overlap between the fields dictate the level of disc. All states in an inter-section have exact same $\mathbf{x}(s)$
    - As a result same estimate value, Smaller regions results in better discrimination

Tile Coding

Computationally efficient way to perform coarse coding
Performs exhaustive partition of space using overlapping grids

TC uses squares arranged in a grid called tiling for state aggregation

- Using squares for state aggregation enables efficient computation
- Increasing the space scaling in a particular direciton results in increased gen. in that particular direction
  - Each dimension of the state space is partitioned by a different set of tiles
    Increase in dimension of state space, results in exponential increase in number of tiles
- To improve disc. several tilings are put on top of each other with a small offset

TC with TD Learning

Recall: $\nu_\pi(s) \approx \widehat{\nu}(s, \mathbf{w}) = \mathbf{w}^T\mathbf{x}(s)$
With TC, $\mathbf{x}(s)$ is a sparse binary vector

In the dot operation, only the weights in non-zero position in $\mathbf{x}(s)$ are multiplied by 1

- To reduce computations, it can be replaced by summation
- Note that TC is a non-linear function over state space

Neural Networks

Parametrized function

A flexible and powerful class of non-linear function approximators

Output of the node 1 in the hidden layer is $f(\mathbf{s}\mathbf{w}_1)$, where $\mathbf{s} = [s_1, s_2]$ $\mathbf{w}_1 = [w_{1,1}, w_{1,2}]^T$, and $f(\cdot) \in \{\tanh(\cdot), \text{relu}(\cdot), \text{step}(\cdot)\}$

- Similarly, outputs from hidden layer are $f(\mathbf{s}\mathbf{W})$, where $\mathbf{W} = [\mathbf{w}_1\, \mathbf{w}_2\, \mathbf{w}_3]$
  - As a result NN is paramaterized function in $\mathbf{W}$

Non-linear functions

NN can be viewed as constructing features
Are non-linear functions of states

Outputs $f(\mathbf{s} \cdot \mathbf{W})$ of NN are called features, where $f(\cdot)$ is a non-linear fn.

- NN has both fixed (# layers, # units) and adjustable (weights) parameters
  - As a result, NN can improve the features by learning from the data
  - Depth of NN facilitates composition and abstraction
- Recall that TC also converts inputs to features
  Params size, number, shape etc are to be fixed aprior

Gradient Descent Training

Stochastic gradient descent can be used to update the parameters of the NN
Backprop can save computation by computing grads starting at the output

Notation:

$\mathbf{s} \to \mathbf{A} \to \Psi \xrightarrow{f_A} \mathbf{x} \to \mathbf{B} \to \theta \xrightarrow{f_B} \widehat{\mathbf{y}}$

$\Psi = \mathbf{s}\mathbf{A}$

$\mathbf{x} \triangleq f_A(\Psi)$

$\theta \triangleq \mathbf{x}\mathbf{B}$

$\widehat{\mathbf{y}} \triangleq f_B(\theta)$

$L(\widehat{y}_k, y_k) \triangleq \frac{1}{2}(\widehat{y}_k - y_k)^2$

Deriving the gradient :

$$\frac{\partial L(\widehat{y}_k, y_k)}{\partial \mathbf{B}_{jk}} = \frac{\partial L(\widehat{y}_k, y_k)}{\partial \widehat{y}_k}\frac{\partial \widehat{y}_k}{\partial \theta_k}\frac{\partial \theta_k}{\partial \mathbf{B}_{jk}} = \delta_k^{\mathbf{B}} \frac{\partial \theta_k}{\partial \mathbf{B}_{jk}} = \delta_k^{\mathbf{B}}\, \mathbf{x}_j,$$

where $\delta_k^{\mathbf{B}} \triangleq \frac{\partial L(\widehat{y}_k, y_k)}{\partial \widehat{y}_k}\frac{\partial f_B(\theta_k)}{\partial \theta_k}$

$$\frac{\partial L(\widehat{y}_k, y_k)}{\partial \mathbf{A}_{ij}} = \delta_k^{\mathbf{B}} \frac{\partial \theta_k}{\partial \mathbf{A}_{ij}} = \delta_k^{\mathbf{B}}\, \mathbf{B}_{jk} \frac{\partial \mathbf{x}_j}{\partial \mathbf{A}_{ij}} = \delta_k^{\mathbf{B}}\, \mathbf{B}_{jk} \frac{\partial f_A(\Psi_j)}{\partial_j}\mathbf{s}_i$$

$$\frac{\partial L(\widehat{y}_k, y_k)}{\partial \mathbf{A}_{ij}} = \delta_j^{\mathbf{A}}\mathbf{s}_i, \text{ where } \delta_j^{\mathbf{A}} = \delta_k^{\mathbf{B}}\, \mathbf{B}_{jk} \frac{\partial f_A(\Psi_j)}{\partial_j}$$

For each (s, y) in D :
- Compute δ_k^B
- $\nabla_B^{jk} = \delta_k^B \mathbf{x}_j$
- $\mathbf{B} = \mathbf{B} - \alpha_B \nabla_B$
- Compute δ_j^A
- $\nabla_A^{ij} = \delta_k^A \mathbf{s}_i$
- $\mathbf{A} = \mathbf{A} - \alpha_A \nabla_A$

Optimization

Initialize weights such that the node outputs have similar variance
Momentum in weight updates can speed up NN optimization

One strategy is $\mathbf{w}_{\text{init}} \sim \frac{\mathcal{N}(0,1)}{\sqrt{n_{\text{inputs}}}}$, results in output of each neuron to be different
Small variance ensure that outputs are in same range

Improve training by update momentum
$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha \nabla_{\mathbf{w}} L(\mathbf{w}_t) + \lambda \mathbf{M}_t$
$\mathbf{M}_{t+1} \leftarrow \lambda \mathbf{M}_t - \alpha \nabla_{\mathbf{w}} L$
learns faster and gets to the stationary target quickly

Control with approx.

Episodic Sarsa

Recall: State-values and Action-values
$\nu_\pi(s) \approx \widehat{\nu}(s, \mathbf{w}) \triangleq \mathbf{w}^T\mathbf{x}(s)$
$q_\pi(s, a) \approx \widehat{q}(s, a, \mathbf{w}) \triangleq \mathbf{w}^T\mathbf{x}(s, a)$

Sarsa needs action-value fns, so feature rep., $\mathbf{x}(s, a)$ needs to include actions as well

Stacked representation:
$\mathbf{x}(s, a) = [x_0(s), x_1(s), x_0(s), x_1(s), \cdots]^T$, where
$\mathbf{x}(s, a_0) = [x_0(s), x_1(s), 0, 0, \cdots]^T$ and
$\mathbf{x}(s, a_1) = [0, 0, x_0(s), x_1(s), \cdots]^T$ and

Loop for each episode:
- $S, A \leftarrow$ initial state and action (ϵ-greedy)
- Loop for each step of episode:
  - Take action A, observe R, S'
  - If terminal:
    - $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R - \widehat{q}(S, A, \mathbf{w})]\nabla\widehat{q}(S, A, \mathbf{w})$
    - Break
  - Choose A' as a fn. of $\widehat{q}(S, \cdot, \mathbf{w})$($\epsilon$-greedy)
  - $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma\widehat{q}(S', A', \mathbf{w}) - \widehat{q}(S, A, \mathbf{w})]\nabla$
  - $S \leftarrow S', A \leftarrow A'$

Expected Sarsa

Recall: Sarsa
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha$ tgt
tgt $\triangleq [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
Expected Sarsa:
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha$ tgt
tgt $\triangleq [R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a') - Q(S_t, A_t)]$

Sarsa (With function approximation)
$\mathbf{w} \leftarrow \mathbf{w} + \alpha$ tgt $\nabla\widehat{q}(S_t, A_t, \mathbf{w})$
tgt $\triangleq [R_{t+1} + \gamma\widehat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \widehat{q}(S_t, A_t, \mathbf{w})]$
Expected Sarsa:
$\mathbf{w} \leftarrow \mathbf{w} + \alpha$ tgt $\nabla\widehat{q}(S_t, A_t, \mathbf{w})$
tgt $\triangleq [R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1})\widehat{q}(S_{t+1}, a', \mathbf{w}) - \widehat{q}(S_t, A_t, \mathbf{w})]$

Q-learning
$\mathbf{w} \leftarrow \mathbf{w} + \alpha$ tgt $\nabla\widehat{q}(S_t, A_t, \mathbf{w})$
tgt $\triangleq [R_{t+1} + \gamma \max_{a'} \widehat{q}(S_{t+1}, a', \mathbf{w}) - \widehat{q}(S_t, A_t, \mathbf{w})]$
With $q_\pi(s, a) \approx \widehat{q}(s, a, \mathbf{w}) = \mathbf{w}^T\mathbf{x}(s, a)$

Exploration

Linear: $q_\pi(s, a) \approx \widehat{q}(s, a, \mathbf{w}) = \mathbf{w}^T\mathbf{x}(s, a)$
Use optimistical initial values
$\mathbf{w} \leftarrow [100\;100\; \cdots]^T$

non-Linear: use ϵ-greedy to choose A_t
$q_\pi(s, a) \approx \widehat{q}(s, a, \mathbf{w}) = \text{NN}(s, a, \mathbf{w})$

Suggestions: mfrahaman_ece@mgit.ac.in, fayazur.mohammad@gmail.com

Dr. Fayazur Rahaman Mohammad,
Dept. of ECE, Mahatma Gandhi Institute of Technology, Hyderabad.

- **Continuing Tasks**
  - **Introduction**
    - **Discount Factor**
      - To ensure that actions maximize reward over time, the discount factor γ needs to be increased towards 1
      - However, larger γ results in larger and variable sums that are difficult to learn
      - Alternatively, with objective of average rewards, agent cares equally about near-by and long-term rewards
  - **Average Reward**
    - **Formulation**
      - $r(\pi) \triangleq \lim_{h\to\infty} \sum_{t=1}^{h} \mathbb{E}[R_t|S_0, A_{0:t-1} \sim \pi]$
        More generally
        $r(\pi) = \sum_s \mu_s(s) \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) r$
        where $\mu_s(s)$ is state visitations
      - $r(\pi)$ helps in determining which policy is better
        But, given a state, how do we choose which action is better
      - Differential returns:
        $G_t = R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \cdots$
        Indicates how much more reward the agents receives compared to the avg return form a policy
      - Differential return is a convergent sum only if it is equal to the true avg reward
        Else it diverges to positive or negative infinity
    - **Differential Value Functions**
      - Value functions differential returns
        $q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)(r - r(\pi) + \sum_{a'} \pi(a'|s') q_\pi(a', s'))$
      - Differential semi-grad Sarsa for estimating $\widehat{q} \approx q_*$
      - Initialize S, A
        Loop for each step:
          Take action A, observe R, S'
          $\delta \leftarrow R - \bar{R} + \widehat{q}(S', A', \mathbf{w}) - \widehat{q}(S, A, \mathbf{w})$
          $\bar{R} \leftarrow \bar{R} + \beta\delta$
          $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla \quad \widehat{q}(S, A, \mathbf{w})$
          $S \leftarrow S', A \leftarrow A'$

Suggestions: mfrahaman_ece@mgit.ac.in, fayazur.mohammad@gmail.com

Learning Policy Directly

Dr. Fayazur Rahaman Mohammad,
Dept. of ECE, Mahatma Gandhi Institute of Technology, Hyderabad.

Learning Policy Directly

Paramatenized Policies

Introduction

All methods so far estimate action-values to learn policies

Every control algo was build on GPI

Policy without Action-Value

Notation:
Policy $\pi(a|s,\boldsymbol{\theta})$ outputs the probability of action a, given state s and parameters $\boldsymbol{\theta}$

Constraints on the Policy Parameterization:
$\pi(a|s,\boldsymbol{\theta}) \geq 0,\ \forall a \in \mathcal{A}$ and $s \in \mathcal{S}$
$\sum_{a\in\mathcal{A}} \pi(a|s,\boldsymbol{\theta}) = 1,\ \forall s \in \mathcal{S}$

The Softmax Policy Parameterization:
$\pi(a|s,\boldsymbol{\theta}) \triangleq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_{b\in\mathcal{A}} e^{h(s,b,\boldsymbol{\theta})}}$,
where $h(s,a,\boldsymbol{\theta})$ is Action Preference

Action preference indicates how much an agent prefers an action
$h(s,a,\boldsymbol{\theta}) = NN(\mathbf{x}(s,a),\boldsymbol{\theta})$
Softmax converts any value of $h()$ to probabilities

Advantages

Exploration of the ϵ-greedy policy would depend on us to specify the amount of exploration

Whereas the parameterized policies can autonomously decrease exploration over time

They can avoid failures due to deterministic policies with limited function approximation

Sometimes the policy is less complicated than the value funciton

They start-off as Stochastic policies and gradually converges to be a Deterministic policy

Learning Policies

Objectives function

Goal is to maximize the reward in the long run:
Episodic: $G_t = \sum_{t=0}^{T} R_t$
Continuing: $G_t = \sum_{t=0}^{\infty} \gamma^t R_t$ (discounted sum)
$G_t = \sum_{t=0}^{\infty} R_t - r(\pi)$ (differential sum)

Average Reward Objective:
$r(\pi) = \sum_s \mu(s) \sum_a \pi(a|s,\boldsymbol{\theta}) \sum_{s',r} p(s',r|s,a) r$
$= \mathbb{E}_\pi[R_t]$

Policy-Gradient Method:
i.e., maximizing the reward objective
$\nabla_\theta r(\pi) = \nabla_\theta \sum_s \mu(s) \sum_a \pi(a|s,\boldsymbol{\theta}) \sum_{s',r} p(s',r|s,a) r$

But, $\mu(s)$ is dependent on $\boldsymbol{\theta}$

Policy Gradient Theorem

It allows us to write Gradient of $r(\pi)$

Gradient of objective is given by:
$\nabla_\theta r(\pi) = \sum_s \mu(s) \sum_a \nabla_\theta \pi(a|s,\boldsymbol{\theta}) q_\pi(s,a)$

This allows us to build an incremental gradient policy update algo using agents experience

Sample based

Estimating Gradient

Getting Stochastic Samples of the Gradient:
$\nabla_\theta r(\pi) = \sum_s \mu(s) \sum_a \nabla_\theta \pi(a|s,\boldsymbol{\theta}) q_\pi(s,a)$
summation over all state is impossible

Instead update paramaters for each visit to a $S_t \sim \pi$
$S_0, A_0, R_1, S_1, A_1, \cdots, S_t, A_t, R_{t+1}, \cdots$
$\boldsymbol{\theta}_{t+1} \triangleq \boldsymbol{\theta}_t + \alpha \sum_a \nabla_\theta \pi(a|S_t,\boldsymbol{\theta}_t) q_\pi(S_t,a)$

Grad. Ascent for policy params:
$\boldsymbol{\theta}_{t+1} \triangleq \boldsymbol{\theta}_t + \alpha \frac{\nabla_\theta \pi(a|S_t,\boldsymbol{\theta}_t)}{\pi(a|S_t,\boldsymbol{\theta}_t)} q_\pi(S_t,A_t)$
$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_\theta \ln[\pi(a|S_t,\boldsymbol{\theta}_t)] q_\pi(S_t,A_t)$

To compute the udpate, need two component:
Grad of the policy and estimate of diff. values

Actor-Critic method for control

Elegently mixes direct policy optimization, value functions and TD learning

Parameter policy setup acts as an Actor
Value function play the role of a Critic evaluating the policy selected by the Actor

Actor and critic learn at the same time, constantly interacting
Actor continously changes the policy to exceed critic's expectation
Critic continously updates the value function to evaluate the actor's changing policy

Using approximate of q_π, we get
$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_\theta \ln[\pi(a|S_t,\boldsymbol{\theta}_t)][R_{t+1} - \bar{R} + \widehat{\nu}(S_{t+1},\mathbf{w})]$

The Critic learns the estimate of $\widehat{\nu}(S_{t+1},\mathbf{w})$ (differential) value function
Average Reward Semi-Gradient TD(0) is used to train Critic

Actor-Critic for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$, select
A differentiable policy param. $\pi(a|s,\boldsymbol{\theta})$
A differentiable state-value fn. param. $\widehat{\nu}(s,\mathbf{w})$

Subtracting baseline $\widehat{\nu}(S_t,\mathbf{w})$ reduces the update variance and thereby enables faster learning
$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_\theta \ln[\pi(a|S_t,\boldsymbol{\theta}_t)]\, \delta_t$
where TD-error $\delta_t = [R_{t+1} - \bar{R} + \widehat{\nu}(S_{t+1},\mathbf{w})]$

Initialize $S \in \mathcal{S}, \bar{R}, \mathbf{w}, \boldsymbol{\theta}, \alpha^{\mathbf{w}}, \alpha^{\boldsymbol{\theta}}, \alpha^{\bar{R}}$
Loop forever (for each time step):
- $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
- Take action A, observe S', R
- $\delta \leftarrow [R_{t+1} - \bar{R} + \widehat{\nu}(S',\mathbf{w}) - \widehat{\nu}(S',\mathbf{w})]$
- $\bar{R} \leftarrow \bar{R} + \alpha^{\bar{R}}\delta$
- $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta \nabla \widehat{\nu}(S,\mathbf{w})$
- $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\delta \nabla \ln[\pi(A|S,\boldsymbol{\theta})]$
- $S \leftarrow S'$

Actor-Critic Specific Implement.

AC with linear function approximation and Softmax policy paramatenization

Policy update with a Softmax Policy:
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\delta \nabla \ln[\pi(A|S,\boldsymbol{\theta})]$
$\pi(a|s,\boldsymbol{\theta}) \triangleq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_{b\in\mathcal{A}} e^{h(s,b,\boldsymbol{\theta})}}$
We have a Softmax policy for each state

Features of the Action Preference Function,
$\widehat{\nu}(s,\mathbf{w}) \triangleq \mathbf{w}^T \mathbf{x}(s)$
$h(s,a,\boldsymbol{\theta}) \triangleq \boldsymbol{\theta}^T \mathbf{x}_n(s,a)$
We use stacked feature vector $\mathbf{x}$

Update rule with Softmax Policy:
$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\, \delta\, \mathbf{x}(S)$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\, \delta\, \mathbf{x}_h(s,a) - \sum_b \pi(b|s,\boldsymbol{\theta}) \mathbf{x}_h(s,b)$

Gaussian Policy

To learn parameters for state dependent Guassian distribution policy for **continuous** action space

Gaussian Distribution:
$p(x) \triangleq \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{(x-\mu)^2}{2\sigma^2})$

Gaussian Policy: (μ, σ are functions of $S, \boldsymbol{\theta}$)
$\pi(a|s,\boldsymbol{\theta}) \triangleq \frac{1}{\sigma(s,\boldsymbol{\theta})\sqrt{2\pi}} \exp(-\frac{(a-\mu(s,\boldsymbol{\theta}))^2}{2\sigma(s,\boldsymbol{\theta})^2})$
$\mu(s,\boldsymbol{\theta}) \triangleq \boldsymbol{\theta}_\mu^T \mathbf{x}(s)$ and $\sigma(s,\boldsymbol{\theta}) \triangleq \exp(\boldsymbol{\theta}_\sigma^T \mathbf{x}(s))$
Note that different μ, σ for each state and
$\boldsymbol{\theta} \triangleq [\boldsymbol{\theta}_\mu\, \boldsymbol{\theta}_\sigma]^T$

Sampling Actions from the Gaussian Policy
Note that σ indicates the degree of exploration in each state
As learning increase, σ is expected to decrease and sampled action is expected to be close to μ

Gradient fo the $\ln(\cdot)$ of the Gaussian Policy
$\nabla \ln \pi(a|s,\boldsymbol{\theta}_\mu) = \frac{1}{\sigma(s,\boldsymbol{\theta})^2}(a - \mu(s,\boldsymbol{\theta}))\mathbf{x}(s)$
$\nabla \ln \pi(a|s,\boldsymbol{\theta}_\sigma) = (\frac{(a-\mu(s,\boldsymbol{\theta}))^2}{\sigma(s,\boldsymbol{\theta})^2} - 1)\mathbf{x}(s)$

Advantages of Continuous Actions:
Might not be straightforward to choose a proper discrete set of actions
Continuous actions allow us to generalize over actions

References:
[1] "Reinforcement Learning An Introduction" by Richard S.Sutton and Andrew G. Barto
[2] "Reinforcement Learning Specialization" by University of Alberta (Coursera online)

Suggestions: mfrahaman_ece@mgit.ac.in, fayazur.mohammad@gmail.com