## Let's go Markov

Markov Process

- Markov property: $P[s_{t+1}|s_t] = P[s_{t+1}|s_1, \dots, s_t]$
- State transition probabilities
  - $P_{ss'} = P[S_{t+1} = s'|S_t = s]$
  - Matrix rows have to sum to 1

Markov Reward Process

- Return: $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$
- Bellman Equation for MRPs
  - $v(s) = E[R_t|S_t = s] = E[r_{t+1} + \gamma v(S_{t+1})|S_t = s]$
- Sum notation: $v(s) = R_s + \gamma \sum_{s' \in S} P_{ss'} v(s')$
- Direct solution
  - $v = R + \gamma P v$
  - $v = (1 - \gamma P)^{-1} R$

Policy: $\pi_t(a, s) = P[A_t = a|S_t = s]$

## Markov Decision Process

Immediate reward: $R^a_{ss'} = r(s, a, s')$

Value function

- $V^\pi(s) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} |S_t = s]$
- $= \mathbb{E}[r_{t+1}|S_t = s] + \gamma \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} |S_t = s]$
- $V^\pi(s) = \sum_{a \in A} \pi(s, a) \sum_{s' \in S} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right)$

Iterative Policy Evaluation Algorithm

- Input π, the policy to be evaluated
- Initialize V(s) = 0, for all s ∈ $S^+$
- Repeat
  - Δ ← 0
  - For each s ∈ S:
    - v ← V(s)
    - $V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]$
  - Δ ← max(Δ, |v - V(s)|)
- until Δ < θ (a small positive number)
- Output V ≈ $V^\pi$

State-Action Value function

- $Q^\pi(s, a) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} |S_t = s, A_t = a]$
- $V^\pi(s) = \sum_{a \in A} \pi(s, a) Q^\pi(s, a)$

Optimal value function

- Optimal State-Action Value function
  - $Q^*(s, a) = \max_\pi Q^\pi(s, a)$
  - $Q^*(s, a) = E[r_{t+1} + \gamma V^*(s_{t+1})|S_t = s, A_t = a]$

Bellman Optimality

- Bellman Optimality Equation for V
  - $V^*(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right)$
- Bellman optimality equation for Q*
  - $Q^*(s, a) = \sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a')\right)$

## Dynamic Programming

Policy Improvement

- Policy Improvement Theorem
  - $Q^\pi(s, \pi'(s)) \geq V^\pi(s) \to V^{\pi'}(s) \geq V^\pi(s)$
- Bellman Optimality Equation (BOE)
  - $V^\pi(s) = \max_{a \in A} Q^\pi(s, a)$

Bellman's Principle of Optimality Theorem: A policy achieves the optimal value iff for any state s' reachable from s, π achieves the optimal value from state s'

Value Iteration Algorithm

- Initialize V arbitrarily, e.g., V(s) = 0, for all s ∈ $S^+$
- Repeat
  - Δ ← 0
  - For each s ∈ S:
    - v ← V(s)
    - $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$
    - Δ ← max(Δ, |v - V(s)|)
- until Δ < θ (a small positive number)
- Output a deterministic policy π such that
  - $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$

Policy Iteration Algorithm

- 1 Initialization
  - V(s)∈R and π(s)∈A(s) arbitrarily for all s∈S
- 2 Policy evaluation
  - Repeat
    - Δ ← 0
    - For each s ∈ S:
      - v ← V(s)
      - $V(s) \leftarrow \sum_{s'} \mathcal{P}^{\pi(s)}_{ss'} \left[\mathcal{R}^{\pi(s)}_{ss'} + \gamma V(s')\right]$
    - Δ ← max(Δ, |v - V(s)|)
  - until Δ < θ (a small positive number)
- 3 Policy improvement
  - Policy-stable ← true
  - For each s ∈ S:
    - b ← π(s)
    - $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$
    - If b ≠ π(s), then policy-stable ← false
  - If policy-stable, then stop; else go to 2

Generalized Policy Iteration Algorithm

- Evaluation: π → V
  - V → V^π
- Improvement: V → π
  - π → greedy(V)

## Model-Free Control

MC Policy Evaluation

- procedure MonteCarloEstimation(π)
  - Init
    - V-hat(s) ← arbitrary value, for all s ∈ S
    - Returns(s) ← an empty list, for all s ∈ S
- EndInit
- repeat
  - Get trace, τ, using π.
  - for all s appearing in τ do
    - R ← return from first appearance of s in τ
    - Append R to Returns(s)
    - V-hat(s) ← average(Returns(s))
- until forever

MC Batch averaging

- $\mu = \frac{1}{k} \sum_j x_j$

MC Online averaging

- $\mu_k = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$

Incremental Monte-Carlo Updates

- $V(s_t) \leftarrow V(s_t) + \frac{1}{N(s_t)} \left(R_t - V(s_t)\right)$
- non-stationary: $V(s_t) \leftarrow V(s_t) + \alpha \left(R_t - V(s_t)\right)$

Monte-Carlo update $V(s_t)$ += $\alpha(R_t - V(s_t))$

Temporal Difference update every time-step

- $V(s_t)$ += $\alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$

Temporal Difference Error: $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$

Temporal Difference Target: $r_{t+1} + \gamma V(s_{t+1})$

TD value function estimation Algorithm

- procedure TD-Estimation(π)
  - Init
    - V-hat(s) <- arbitrary value, for all s ∈ S
  - EndInit
  - repeat(For each episode)
    - Initialize s
    - repeat(For each step of episode)
      - a action chosen from t at s
      - Take action a; observe r, and next state, s'
      - δ <- r + γV-hat(s) - V-hat(s)
      - V-hat(s) <- V-hat(s) + αδ
      - s <- s'
    - until s is absorbing state
  - until Done
- end procedure

Greedy policy improvement

- $\pi'(s) = \arg\max_{a \in A} \mathcal{R}^{s'}_{sa} + \mathcal{P}^a_{ss'} V(s')$
- π'(s) = $\text{argmax}_a$ Q(s, a)

On-Policy

- Soft control, ϵ-greedy, SARSA

Off-Policy

- Importance sampling, Q-Learning

ϵ-greedy policy with ϵ ∈ [0, 1]

- π(s, a) =
- 1 − ϵ + ϵ/|A(s)|, if a = a* = $\text{argmax}_a$ Q(s, a)
- ϵ/|A(s)|, if a ≠ a*

On-policy ϵ-greedy first-visit MC control algorithm

- Code
  - Initialize, for all s ∈ S, a ∈ A(s):
    - Q(s,a) ← arbitrary
    - Returns(s,a) ← empty list
    - π(a|s) ← an arbitrary ϵ-soft policy
  - Repeat forever:
    - (a) Generate an episode using π
    - (b) For each pair s, a appearing in the episode:
      - G ← return following the first occurrence of s, a
      - Append G to Returns(s,a)
      - Q(s,a) ← average(Returns(s,a))
    - (c) For each s in the episode:
      - a* ← $\text{argmax}_a$ Q(s,a)
      - For all a ∈ A(s):
        - π(a|s) ←
          - 1 − ϵ + ϵ/|A(s)|, if a = a*
          - ϵ/|A(s)|, if a ≠ a*

Greedy in the Limit with Infinite Exploration (GLIE)

- All state-action pairs are explored infinitely many times,
  - $\lim_{k\to}$ $N_k(s, a) = \infty$
- The policy converges on a greedy policy,
  - $\lim_{k\to}$ $\pi_k(a, s)$
    - = (a == $\text{argmax}_{a'}$ $Q_k(s, a')$)
  - '==' evaluates to 1 if true and 0 else
- e.g. if ϵ reduces to zero with $\epsilon_k = 1/k$

MC Batch Learning to Control

- procedure MonteCarloBatchOptimization(n)
  - Init
    - Q-hat(s, a) <- arbitrary value, for all s∈S, a∈A.
    - π <- ϵ-greedy(Q-hat)
  - EndInit
  - repeat (For each batch)
    - Returns(s, a) <- an empty list, for all s∈S
    - for i = 1 to n do
      - Get trace, τ, using π
      - for all (s,a) appearing in T do
        - R <- return from first appearance of (s, a) in τ.
        - Append R to Returns(s, a)
    - for all s∈S, a∈A do
      - Q-hat(s, a) <- average(Returns(s, a))
    - π <- ϵ-greedy(Q-hat)
  - until Done
  - return greedy(Q-hat)

MC Iterative Learning to Control algorithm

- procedure MonteCarloIterativeOptimization(n)
  - Init
    - Q-hat(s, a) <- arbitrary value, for all s∈S, a∈A.
    - π <- ϵ-greedy(Q-hat)
  - EndInit
  - for i = 1 to n do
    - Get trace, τ, using π
    - for all (s,a) appearing in T do
      - R <- return from first appearance of (s, a) in τ.
      - Q-hat(s, a) <- Q-hat(s_t, a_t) + α[R - Q-hat(s_t, a_t)]
    - π <- ϵ-greedy(Q-hat)
  - return greedy(Q-hat)

Sarsa: Q(S,A) ← Q(S,A) + α(r + γQ(S',A') − Q(S, A))

Sarsa algorithm

- Initialize Q(s,a), ∀s∈S, a∈A(s), arbitrarily, and Q(terminal-state, ·) = 0
- Repeat (for each episode):
  - Initialize S
  - Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  - Repeat (for each step of episode):
    - Take action A, observe R, S'
    - Choose A' from S' using policy derived from Q (e.g, e-greedy)
    - Q(S,A) ← Q(S,A) + α(R + γQ(S', A') − Q(S, A))
    - S ← S'; A←A';
  - until S is terminal

Robbins-Munro sequence of step-sizes $\alpha_t$:

- $\sum_{t=1}^{\infty}\alpha_t = \infty \wedge \sum_{t=1}^{\infty}\alpha_t^2 < \infty$
- e.g. αt = α/t for α > 0

Q-Learning:

- Q(S, A) <- Q(S, A) + α(R + γ $\max_{a'}$ (S', a') - Q(S, A))

Q-Learning algorithm

- Initialize Q(s,a), ∀s∈S, a∈A(s), arbitrarily, and Q(terminal-state, ·) = 0
- Repeat (for each episode):
  - Initialize S
  - Repeat (for each step of episode):
    - Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    - Take action A, observe R, S'
    - Q(S,A) ← Q(S,A) + α[R + γ max_a Q(S', a) − Q(S, A)]
    - S ← S';
  - until S is terminal

| | Full Backup (DP) | Sample Backup (TD) |
|---|---|---|
| **Bellman Expectation Equation for v_π(s)** | Iterative policy evaluation V(s) ← E[R + γV(S') \| s] | TD Learning V(s) ←α R + γV(S') |
| **Bellman Expectation Equation for q_π(s, a)** | Q-Policy Iteration Q(s,a) ← E[R + γQ(S',A') \| s,a] | Sarsa Q(S,A) ←α R + γQ(S',A') |
| **Bellman Optimality Equation for q*(s, a)** | Q-Value Iteration Q(s,a) ← E[R + γ max_a' Q(S',a') \| s,a] | Q-Learning Q(S,A) ← R + γ max_a' Q(S',a') |

## Function Approximation

Estimate value function with function approximation

- V^π(s) ≈ V-hat(s, w)
- Q^π(s, a) ≈ Q-hat(s, a, w)

Stochastic Gradient Descent

- J(w) = E[$V^{\pi}$(s) - V-hat(s,w)$^2$]
- Δw = -1/2 α$\nabla_w$J(w)
  - = α($V^{\pi}$(s) - V-hat(s, w))$\nabla_w$V-hat(s, w)

Represent state by a feature vector

- x(s) = ($x_1$(s) ... $x_n$(s))$^T$
- v-hat(s, w) = x(s)$^T$w

Radial Basis Functions: $\phi_s(i) = \exp\left(-\frac{|s-c_i|^2}{2\sigma_i^2}\right)$

Update rule: Δw = α($V^{\pi}$(s) - V-hat(s,w))x(s)

- $\nabla_w$V-hat(s,w) = x(s)

Value function estimation using linear function approximation

- Initialize w = 0, k = 1
- loop
  - Sample tuple (s_k, a_k, k, s_(k+1)) given π
  - Update weights:
    - w = w + a(r + γx(s)^T w - x(s)^T w)x(s)
  - k = k + 1
- end loop

## Deep Q Learning

Q-Learning vs Lin FA Q-Learning

- Initialize Q(s,a), ∀s∈S, a∈A(s), arbitrarily, and Q(terminal-state, ·) = 0
- Repeat (for each episode):
  - Initialize S
  - Repeat (for each step of episode):
    - Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    - Take action A, observe R, S'
    - Q(S,A) ← Q(S,A) + α[R + γ max_a Q(S', a) − Q(S, A)]
    - S ← S';
  - until S is terminal

DQN TD error(w) $= r_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'; w) - Q(s_t, a; w)$

Δw = α $\left[r + \gamma \max_{a'} Q(S_{t+1}, a'; w) - Q(s_t, a; w)\right] \nabla_w Q(S_t, a; w)$

DQN algorithm

- Initialize replay memory D to capacity N
- Initialize action-value function Q with random weights
- for episode = 1, M do
  - Initialize state $s_t$
  - for t = 1, T do
    - With probability ϵ select a random action a_t
    - otherwise select $a_t$ = $\max_a$ Q*($s_t$, a; θ)
    - Execute action at and observe reward $r_t$ and state $s_{t+1}$
    - Store transition ($s_t$, $a_t$, $r_t$, $s_{t+1}$) in D
    - Set $s_{t+1}$ = $s_t$
    - Sample random minibatch of transitions ($s_t$, $a_t$, $r_t$, $s_{t+1}$) from D
    - Set $y_j$ =
      - $r_j$ for terminal $s_{t+1}$
      - $r_j$ + γ $\max_{a'}$ Q($s_{t+1}$, a'; θ) for non-terminal $s_{t+1}$
    - Perform a gradient descent step on ($y_j$ - ($s_t$, $a_j$; θ))$^2$
  - end for
- end for

DDQN: $Q^*(s_t, a_t) \approx r_t + \gamma Q'\left(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a')\right)$

## Tricks of the Trade

Formatting inputs

- Frame Stacking
- Input normalization
- Frames of reference/Coordinate Systems

Control of training

- Bootstrapping
- Reward normalization or Reward standardization
- Target networks
- Weight initialization
- Gradient clipping
- Considering (mini-)batch sizes
- Revisiting continuous actions

## Policy Gradients

$p_\tau$(θ) = $p_\theta$($s_1$, $a_1$, ..., $s_T$, $a_T$)

$p_\tau(\theta) = p(s_1)\prod_{t=1}^{T}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$

Infinite horizon: $\theta^* = arg\ max_\theta\ \mathbb{E}\left[\sum_{t=1}^{T} r(s_t, a_t)\right]_{\tau \propto p_\theta(\tau)}$

Finite horizon: $\theta^* = \arg\max_\theta \sum_{t=0}^{T} \mathbb{E}[r(s_t, a_t)]_{(s_t,a_t)\sim p_\theta(\tau)}$

Finite Difference Policy Gradient update rule:

- $\theta_k$ += α((J(θ + $u_k$ϵ) − J(θ)) / ϵ)

Direct Policy gradients

- $\theta^* = \arg\max_\theta \mathbb{E}\left[\sum_t r(s_t, a_t)\right]_{\tau\sim p_\theta(\tau)}$
- $J(\theta) = \mathbb{E}\left[\sum_t r(s_t, a_t)\right]_{\tau\sim p_\theta(\tau)}$

  $\approx \frac{1}{N}\sum_{i=1}^{N}\sum_t r(s_{i,t}, a_{i,t})$
- J(θ) = $\sum_s$ $p^{\pi}$(s)$V^{\pi}$(s)

$\nabla_\theta J(\theta)$

$= E_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_{t=1}^{T} r(s_t, a_t)\right)\right]$

REINFORCE algorithm:

- Sample {τ^i} from π_θ(a_t|s_t) (run the policy)
- ∇_θ J(θ) ≈
  - $\sum_i\left(\sum_t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\right)\left(\sum_t r(s_t^i, a_t^i)\right)$
- θ <- θ + α∇_θ J(θ)

Gaussian Distribution

- p(x|μ,∑) = (2π)$^{-D/2}$|∑|$^{-1/2}$exp(−1/2 (x − μ)$^T$∑$^{-1}$(x − μ))
- $\nabla_\theta J(\theta)$

  $\approx \frac{1}{N}\sum_i^N\left(\sum_t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\right)\left(\sum_t r(s_t^i, a_t^i)\right)$

Example: πθ(at|st) = N(fneural network(st);∑)

- $\nabla_\theta \log \pi_\theta(a_t|s_t) = -\frac{1}{2}\Sigma^{-1}(f(s_t) - a_t)\frac{\mathrm{d}f}{\mathrm{d}\theta}$

Maximum likelihood

- $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log \pi_\theta(\tau_i)\, r(\tau_i)$
- $\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log \pi_\theta(\tau_i)$

## Actor-Critic Methods

Actor: policy-based learning

Critic: value-based learning

Advantage Actor-Critic (A2C) Algorithm

- Sample {s_i, a_i} from π_θ(a|s) (run it on the robot)
- Fit $\hat{V}_\phi^\pi(s)$ to sampled reward sums
  - $y_{i,t} \approx r(s_{i,t}, a_{i,t}) + \hat{V}_\phi^\pi(s_{i,t+1})$
  - $\mathcal{L}(\phi) = \frac{1}{2}\sum_i \left\|\hat{V}_\phi^\pi(s_i) - y_i\right\|^2$
- Evaluate $\hat{A}^\pi(s_i, a_i) = r(s_i, a_i) + \hat{V}_\phi^\pi(s_i') - \hat{V}_\phi^\pi(s_i)$
- $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(a_t|s_t)\, \hat{A}^\pi(s_i, a_i)$
- θ ← θ + α ∇_θ J(θ)

Advantage element: Q(s, a) = V (s) + A(s, a)