Reinforcement Learning: Learning to Act

Ngô Quốc Hưng

COTAI

hung.ngo@cot.ai | FB: curiousAI

Past: AINovation, MLR Stuttgart, IDSIA

VTCA-COTAI AI Foundations for Practitioner - Week 7

Giới Thiệu

Reinforcement Learning (RL) là ngành áp dụng các kỹ thuật học máy vào lĩnh vực điều khiển tối ưu (optimal control). Mô hình RL thường bao gồm các tác nhân (agents) và môi trường (environments). RL tập trung giải quyết vấn đề hành động ra sao (acting, decision making) ở mỗi thời điểm để thay đổi trạng thái (state) của môi trường từ đó đạt được kết quả tối ưu khi kết thúc. Do đó ta tạm dịch RL là "học máy điều khiển".

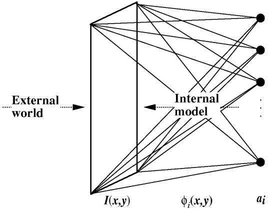
Ứng dụng của RL cực kỳ đa dạng, trong bất cứ lĩnh vực nào cần ra quyết định, trải rộng từ ngành robotics đến ngành quảng cáo trên mạng. Một số ứng dụng "kool ngầu":
– Robot tự học (e.g., Google AI)
– Xe tự hành (deepRL for autonomous driving cars)
– Quản lý tiêu thụ điện (Google HVAC)
– Hệ thống gợi ý (recommender systems), ví dụ trong quảng cáo: đặt panels sao cho xác suất người dùng click lớn nhất (contextual bandit problems), trong hệ thống newsfeed ta đọc hàng ngày.
– Hệ thống hỏi đáp visual question answering (VQA), hệ thống tự sinh hội thoại (deep RL for chatbots, e.g., Google Duplex), tóm tắt văn bản (summarization, e.g., Salesforce)
– Vô địch cờ vây (AlphaGo Zero) và các computer games (e.g., DQN).
– Tự sinh các mạng neuron để giải quyết các bài toán ML (autoML, neural architecture search NAS)
– Tự đặt lệnh mua bán chứng khoán (JPMorgan)

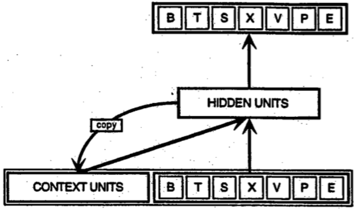
Trong guest lecture này anh Ngô Quốc Hưng (fb.com/curiousAI) sẽ giới thiệu những khái niệm cơ bản của RL, cherry-pick các giải thuật phổ biến, và giới thiệu một số ứng dụng thú vị trong nghiên cứu của tác giả cũng như của các nhóm nghiên cứu khác. (Lưu ý các slides có animations và embedded links đến các bài báo khoa học liên quan, nên mở bằng Adobe Acrobat).

Tài liệu tham khảo (nhập môn): sách free!!! của R. Sutton & A. Barto (**bible**), C. Szepesvari (more math), S. M. LaValle (planning & robotics), J. Norris (Markov chains), và video lectures của D. Silver.

Sequential decision making & world state

Embedding coordinates as (latent) state variables.





Olshausen, 1997

Elman's simple RNN, 1990

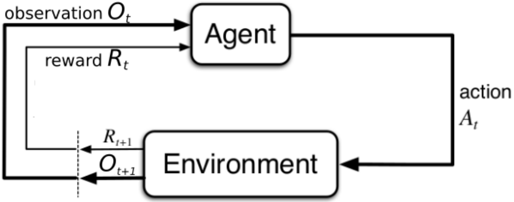
Coordinates of latent embedding as *state* variables $s^{\star}$

Nội dung chính (Outline)

Problem formulation

Learning in MDPs

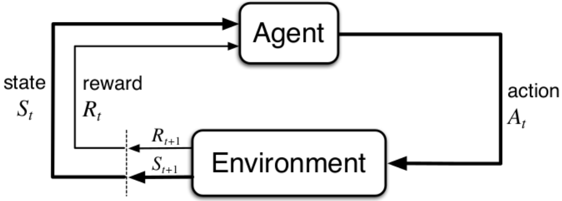
Agent, Environment & Action–Perception Cycle



Consider discrete time steps $t = 0, 1, 2, \ldots$

- At each time step t: Agent makes a "raw" **observation** O_t (e.g., an image), chooses to perform an **action** A_t, and gets an *immediate* **reward** $R_t \in \mathbb{R}$ (a *scalar*).
- **History** $H_t := (O_0, A_0, R_1, O_1, A_1, R_2, O_2, \ldots, O_t)$.
- **Objective**: learn to act *optimally* – which action to choose in a given situation (a history of interactions) in order to maximize *long-term* rewards.

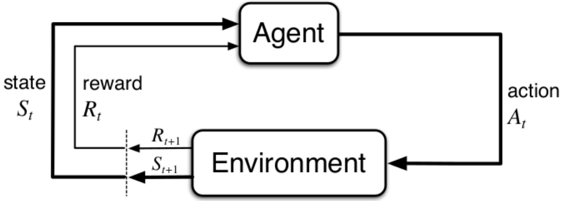
Agent, Environment & Action–Perception Cycle



What is a **state**?

- **State** $S_t = f(H_t)$: a compact, *useful* summary of the history.
- DeepRL: "deep function" f
  - squeezing as much information of the past as possible.
  - example: f = RNN/LSTM & S_t = sequence embedding.

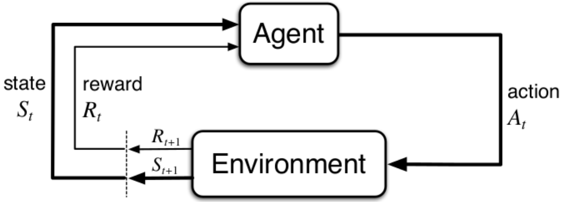
Markov Decision Processes (MDPs)



Simplest **model**? An MDP

- **Markov** state: $Pr(S_{t+1}, R_{t+1} | H_t, A_t) = Pr(S_{t+1}, R_{t+1} | S_t, A_t)$
  - MDP state = *sufficient statistic* of the past (as useful as the actual history for predicting the future.)
  - MDP state = *fully observable*: the state estimated by the agent is exactly the state of the environment.

Agent, Environment & Action–Perception Cycle



Where do **reward signals** R_t come from?

- Specified from the task (designed by domain experts).
  - Often *sparse, delayed*. Sometimes *very* difficult to specify!
  - Reward hypothesis: "Any task can be described by the maximization of expected cumulative reward."
- Self-generated by the agent (curious RL)
- Hybrid (combining both, for balancing exploration–exploitation)
- Hidden (learning from demonstrations, LfD)

Example: Recycling Robot

Recycling robot MDP (Sutton & Barto)

- At each step, robot has a choice of three actions:
  – go out and search for a can
  – wait till a human brings it a can
  – go to charging station to recharge
- Searching is better (higher reward), but runs down battery. Running out of battery power is very bad and robot needs to be rescued
- Decision based on current state – is energy high or low

Example: Recycling Robot

Recycling robot MDP (Sutton & Barto)

- At each step, robot has a choice of three actions:
  – go out and search for a can
  – wait till a human brings it a can
  – go to charging station to recharge
- Searching is better (higher reward), but runs down battery. Running out of battery power is very bad and robot needs to be rescued
- Decision based on current state – is energy high or low
- How would you design reward function for this robot's task?

Example: Recycling Robot

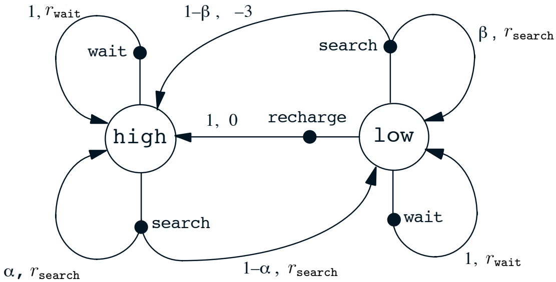
Recycling robot MDP (Sutton & Barto)

- At each step, robot has a choice of three actions:
  – go out and search for a can
  – wait till a human brings it a can
  – go to charging station to recharge
- Searching is better (higher reward), but runs down battery. Running out of battery power is very bad and robot needs to be rescued
- Decision based on current state – is energy high or low
- How would you design reward function for this robot's task?
  One reasonable solution: reward is number of cans (expected to be) collected, negative reward for needing rescue.

Example: Recycling Robot

Transition graph

$$\mathcal{S} = \{\text{high}, \text{low}\}$$
$$\mathcal{A}(\texttt{high}) = \{\text{search}, \text{wait}\}$$
$$\mathcal{A}(\texttt{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$$
$$\mathcal{R} = \{r_{\text{search}}, r_{\text{wait}}, 0, -3\}$$



Next state and reward depend only on current state and action (MDP).

Example: Recycling Robot

This is a Markov decision process (MDP) with a **model**:

$$\text{Transition function } p(s' \mid s, a) := \Pr\left\{S_{t+1} = s' \mid S_t = s, A_t = a\right\}$$
$$\text{Reward function } r(s, a, s') := \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s']$$

Tabular representation, with *unknown* transition/dynamics model:

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1-\alpha$	$r_{\texttt{search}}$
low	search	high	$1-\beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	$r_{\texttt{wait}}$
low	wait	high	0	$r_{\texttt{wait}}$
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	$0.$

Learning to Act: Optimality Criteria

How to formulate "acting optimally"?

- **Policy** π: probability of taking action $a \in \mathcal{A}$ at state $s \in \mathcal{S}$
  – Stochastic policy: $a \sim \pi(a|s) = \Pr\{A_t = a | S_t = s\}$.
  – Deterministic policy: $a = \pi(s)$; a special case of $\pi(a|s)$.
- **Return** $G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1}$

  Sum of *discounted* future rewards in 1 *trajectory/episode/rollout* (pick a policy, execute until termination).

  Discount factor $\gamma \in [0,1]$ emphasizes the recency of rewards & unifying finite/infinite horizon settings.
- **State value function** $v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$

  Expected return when repeatedly follow a policy π from state $s \approx$ empirical mean $\frac{1}{K}\sum_{k=1}^{K} G_t^k$.
- **Action value function** $q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$

  Expected return when repeatedly taking action a from state s then following a policy π.
- **Terminal states** s_g: $v_\pi(s_g) = q_\pi(s_g, a) = 0\ \forall a$ (by convention).
- Find an **optimal policy** π_* that maximizes values at *all* states

$$\forall s \in \mathcal{S}, a \in \mathcal{A}: \ q_*(s,a) := \max_\pi q_\pi(s,a), \ \ v_*(s) := \max_a q_*(s,a)$$

Interesting extensions: *goal-conditioned* reward/policy/value functions $r(s,a,s',g)$, $\pi(a|s,g)$, $q_\pi(s,a,g)$.

Nội dung chính (Outline)

Problem formulation

Learning in MDPs

Solving Markov Decision Processes (MDPs)

How to find an optimal policy?

- Complete (known) MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$: planning problems.
  - Global (offline) planning methods (for all states): VI, PI.
  - Too big? Forward search (local, from "current" state): e.g., MPC, MCTS. Environment $\Leftrightarrow$ simulator!
  - Still too big (e.g., Go game, warehouse/logistics domains)? Learning+planning, e.g., AlphaGo Zero
- Incomplete MDP $\langle \mathcal{S}, \mathcal{A}, \cdot, \cdot, \gamma \rangle$ (unknown model $\{\mathcal{T}, \mathcal{R}\}$):
  - Learning from interactions: agent chooses actions
  - Learning from demonstrations: "expert" chooses actions
- Evaluation vs. Control problems
  - A fixed policy π is given: learning to *predict* value v_π
  - Improve policies iteratively: learning to *control*.

Planning in discrete MDPs

- Complete (known) MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$: *planning* methods.
- Bellman's Optimality Equations:

(Homework: derive these equations from previous definitions of value functions)

$$v_*(s) = \max_a [R(a,s) + \gamma \sum_{s'} P(s'|a,s)\ v_*(s')] \tag{1}$$

$$q_*(s,a) = R(a,s) + \gamma \sum_{s'} P(s'|a,s)\ v_*(s') \tag{2}$$

$$= R(a,s) + \gamma \sum_{s'} P(s'|a,s)\ \max_{a'} q_*(s',a') \tag{3}$$

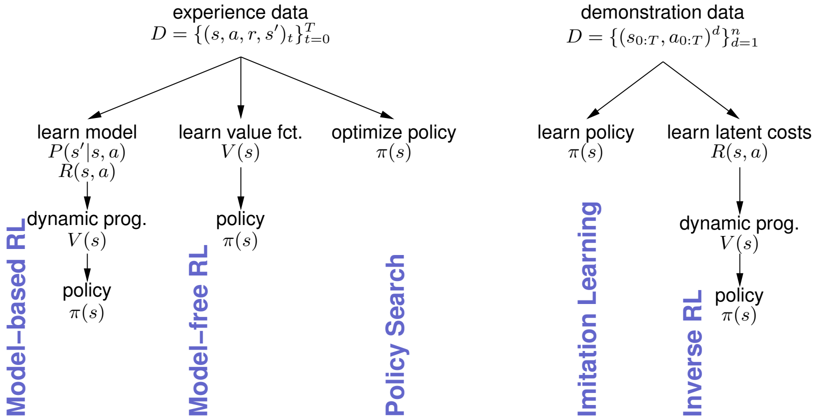
- Optimal policy: *greedy* w.r.t. value functions

$$\pi_*(s) = \operatorname{argmax}_a q_*(s,a) = \operatorname{argmax}_a [R(a,s) + \gamma \sum_{s'} P(s'|a,s)\ v_*(s')]$$

Dynamic Programming turns these equations into iterative update rules (**value/policy iteration**, VI/PI)

Learning to Act: 5 Main Approaches

- Learn from interactions $\mathcal{D} = \{(s, a, s', r)_t\}$
- Learn from demonstrations $\mathcal{D} = \{(s_{0:T}, a_{0:T})_d\}$
  Agent is given demonstrated (e.g., expert's) actions $a_{0:T}$ for each trajectory $s_{0:T}$ instead of reward signals.



Learning from Interactions in MDPs

- Learn from experience $\mathcal{D} = \left\{ \{(s, a, s', r)_t\}_{t=0}^{T_i} \right\}_{i=1}^{k}$

experience
$\{(s, a, r, s')\}$

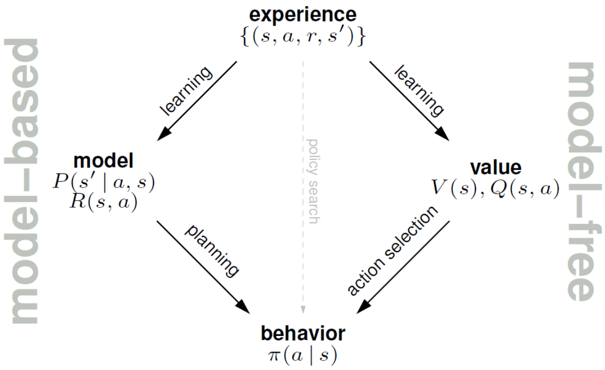
model
$P(s' \mid a, s)$
$R(s, a)$

value
$V(s), Q(s, a)$

behavior
$\pi(a \mid s)$

Learning from Interactions in MDPs

- Learn from experience $\mathcal{D} = \left\{ \{(s, a, s', r)_t\}_{t=0}^{T_i} \right\}_{i=1}^{k}$



Learning from Interactions in MDPs

- Incomplete MDP $\langle \mathcal{S}, \mathcal{A}, \cdot, \cdot, \gamma \rangle$ (unknown model $\{\mathcal{T}, \mathcal{R}\}$).
- Accumulating experience while interacting with the world

$$\mathcal{D} \leftarrow \mathcal{D} \cup \{e_t := (s, a, s', r)_t\}$$

Learning from Interactions in MDPs

- Incomplete MDP $\langle \mathcal{S}, \mathcal{A}, \cdot, \cdot, \gamma \rangle$ (unknown model $\{\mathcal{T}, \mathcal{R}\}$).
- Accumulating experience while interacting with the world

$$\mathcal{D} \leftarrow \mathcal{D} \cup \{e_t := (s, a, s', r)_t\}$$

- What could the RL agent learn from the data?
  – learn to predict next state: $P(s'|s, a)$
  – learn to predict immediate reward: $P(r|s, a, s')$
  $\Rightarrow$ learn a model (essentially *supervised* learning) then *plan*.

  – learn to predict *value*: $s, a \mapsto \hat{q}(s, a) \Rightarrow$ this lecture.
  – learn to predict *action*: $\pi(a|s)$)
  – learn to *control*: $\pi_*(s)$
  (These are hard: sparse, delayed feedback; explore vs. exploit)

Learning from Interactions in MDPs

Learning to predict value $\hat{v}_\pi(s;\theta)$, $\hat{q}_\pi(s,a;\theta)$:

1. Assume true value functions $v_\pi(s)$, $q_\pi(s,a)$ were *known*:
   – Cast as a *regression* (supervised) problem!

$$s \mapsto \hat{v}(s;\theta) \qquad \text{or } s, a \mapsto \hat{q}(s,a;\theta)$$

   Note: these methods include exact (tabular) methods as a special case.

2. Approximate true targets using *estimates* $\tilde{v}_\pi(s)$, $\tilde{q}_\pi(s,a)$:
   – Monte-Carlo (MC) methods use *actual return* as target:

$$\tilde{v}_\pi(s_t) = G(s_t)$$

   – Temporal Difference (TD) methods use *estimated return*:

$$\tilde{v}_\pi(s_t) = r_{t+1} + \gamma\hat{v}(s_{t+1};\theta)$$

$$\text{SARSA: } \tilde{q}_\pi(s_t,a_t) = r_{t+1} + \gamma\hat{q}(s_{t+1},a_{t+1};\theta)$$

$$\text{Q-learning: } \tilde{q}_\pi(s_t,a_t) = r_{t+1} + \gamma \max_{a'} \hat{q}(s_{t+1},a';\theta)$$

Learning Approximate Value Functions

1. Supervised Learning Formulation

- Given training data: $\mathcal{D} = \left\{ \{e_t = (s, a, s', r)_t\}_{t=0}^{T_i} \right\}_{i=1}^{k}$ sampled from some distribution $P(\cdot)$
- Regression problem: minimizing the mean-squared error (similarly for $Q^\pi(s, a)$)

$$L(\theta) = \mathbb{E}_{s \sim P(\cdot)} \left[\left(V^\pi(s) - \hat{V}(s; \theta) \right)^2 \right] \approx \mathbb{E}_{s \sim \mathcal{D}} \left[\left(V^\pi(s) - \hat{V}(s; \theta) \right)^2 \right]$$

Learning Approximate Value Functions

1. Supervised Learning Formulation

- Given training data: $\mathcal{D} = \left\{ \{e_t = (s, a, s', r)_t\}_{t=0}^{T_i} \right\}_{i=1}^{k}$ sampled from some distribution $P(\cdot)$
- Regression problem: minimizing the mean-squared error (similarly for $Q^\pi(s, a)$)

$$L(\theta) = \mathbb{E}_{s \sim P(\cdot)} \left[\left(V^\pi(s) - \hat{V}(s; \theta) \right)^2 \right] \approx \mathbb{E}_{s \sim \mathcal{D}} \left[\left(V^\pi(s) - \hat{V}(s; \theta) \right)^2 \right]$$

- Critical points: $\nabla_\theta L(\theta) = 0 \Leftrightarrow \sum_{s \in \mathcal{D}} \left(V^\pi(s) - \hat{V}(s; \theta) \right) \nabla_\theta \hat{V}(s; \theta) = 0$

Learning Approximate Value Functions

1. Supervised Learning Formulation

- Given training data: $\mathcal{D} = \left\{ \{e_t = (s, a, s', r)_t\}_{t=0}^{T_i} \right\}_{i=1}^{k}$ sampled from some distribution $P(\cdot)$
- Regression problem: minimizing the mean-squared error (similarly for $Q^\pi(s, a)$)

$$L(\theta) = \mathbb{E}_{s\sim P(\cdot)}\left[\left(V^\pi(s) - \hat{V}(s;\theta)\right)^2\right] \approx \mathbb{E}_{s\sim\mathcal{D}}\left[\left(V^\pi(s) - \hat{V}(s;\theta)\right)^2\right]$$

- Critical points: $\nabla_\theta L(\theta) = 0 \Leftrightarrow \sum_{s\in\mathcal{D}} \left(V^\pi(s) - \hat{V}(s;\theta)\right)\nabla_\theta \hat{V}(s;\theta) = 0$
  – Linear functions: analytical, global optimum solution. E.g., linear least squares methods (LSTD, LSPI)

Learning Approximate Value Functions

1. Supervised Learning Formulation

- Given training data: $\mathcal{D} = \left\{ \{e_t = (s, a, s', r)_t\}_{t=0}^{T_i} \right\}_{i=1}^{k}$ sampled from some distribution $P(\cdot)$
- Regression problem: minimizing the mean-squared error (similarly for $Q^\pi(s, a)$)

$$L(\theta) = \mathbb{E}_{s \sim P(\cdot)} \left[\left(V^\pi(s) - \hat{V}(s; \theta) \right)^2 \right] \approx \mathbb{E}_{s \sim \mathcal{D}} \left[\left(V^\pi(s) - \hat{V}(s; \theta) \right)^2 \right]$$

- Critical points: $\nabla_\theta L(\theta) = 0 \Leftrightarrow \sum_{s \in \mathcal{D}} \left(V^\pi(s) - \hat{V}(s; \theta) \right) \nabla_\theta \hat{V}(s; \theta) = 0$
  – Linear functions: analytical, global optimum solution. E.g., linear least squares methods (LSTD, LSPI)
- Gradient descent (GD) solution: batch (offline), SGD (incremental/online), mini-batch (hybrid)
  – SGD (stochastic gradient descent) *samples* the (full) gradient at each s or (s, a)

Learning Approximate Value Functions

1. Supervised Learning Formulation

- Given training data: $\mathcal{D} = \left\{ \{e_t = (s, a, s', r)_t\}_{t=0}^{T_i} \right\}_{i=1}^{k}$ sampled from some distribution $P(\cdot)$
- Regression problem: minimizing the mean-squared error (similarly for $Q^\pi(s, a)$)

$$L(\theta) = \mathbb{E}_{s \sim P(\cdot)} \left[\left(V^\pi(s) - \hat{V}(s; \theta) \right)^2 \right] \approx \mathbb{E}_{s \sim \mathcal{D}} \left[\left(V^\pi(s) - \hat{V}(s; \theta) \right)^2 \right]$$

- Critical points: $\nabla_\theta L(\theta) = 0 \Leftrightarrow \sum_{s \in \mathcal{D}} \left(V^\pi(s) - \hat{V}(s; \theta) \right) \nabla_\theta \hat{V}(s; \theta) = 0$
  – Linear functions: analytical, global optimum solution. E.g., linear least squares methods (LSTD, LSPI)
- Gradient descent (GD) solution: batch (offline), SGD (incremental/online), mini-batch (hybrid)
  – SGD (stochastic gradient descent) *samples* the (full) gradient at each s or (s, a)
  – Least mean squares (LMS), aka Widrow-Hoff rule: SGD update

$$\theta_{i+1} = \theta_i + \alpha_i \left(V^\pi(s) - \hat{V}(s; \theta_i) \right) \nabla_{\theta_i} \hat{V}(s; \theta_i)$$

Learning Approximate Value Functions

1. Supervised Learning Formulation

- Given training data: $\mathcal{D} = \left\{ \{e_t = (s, a, s', r)_t\}_{t=0}^{T_i} \right\}_{i=1}^{k}$ sampled from some distribution $P(\cdot)$
- Regression problem: minimizing the mean-squared error (similarly for $Q^\pi(s, a)$)

$$L(\theta) = \mathbb{E}_{s \sim P(\cdot)} \left[\left(V^\pi(s) - \hat{V}(s; \theta) \right)^2 \right] \approx \mathbb{E}_{s \sim \mathcal{D}} \left[\left(V^\pi(s) - \hat{V}(s; \theta) \right)^2 \right]$$

- Critical points: $\nabla_\theta L(\theta) = 0 \Leftrightarrow \sum_{s \in \mathcal{D}} \left(V^\pi(s) - \hat{V}(s; \theta) \right) \nabla_\theta \hat{V}(s; \theta) = 0$
  – Linear functions: analytical, global optimum solution. E.g., linear least squares methods (LSTD, LSPI)
- Gradient descent (GD) solution: batch (offline), SGD (incremental/online), mini-batch (hybrid)
  – SGD (stochastic gradient descent) *samples* the (full) gradient at each s or (s, a)
  – Least mean squares (LMS), aka Widrow-Hoff rule: SGD update

$$\theta_{i+1} = \theta_i + \alpha_i \left(V^\pi(s) - \hat{V}(s; \theta_i) \right) \nabla_{\theta_i} \hat{V}(s; \theta_i)$$

- Learned behaviors are often unstable/diverge when θ is a nonlinear (e.g., deep convolutional) function ☹
  – *Nonstationary* distribution: when θ is updated $\rightarrow$ policy/behavior & data distribution $P(\cdot)$ changed!
  – *Non-i.i.d.* training data distribution: samples are correlated (generated by interaction, in trajectory)
  – *Unstable* training due to correlations between $\hat{V}(s; \theta)$ and its regression target values (which depend on θ; see next slide).

Learning Approximate Value Functions

1. Supervised Learning Formulation

- Given training data: $\mathcal{D} = \left\{\{e_t = (s, a, s', r)_t\}_{t=0}^{T_i}\right\}_{i=1}^{k}$ sampled from some distribution $P(\cdot)$
- Regression problem: minimizing the mean-squared error (similarly for $Q^\pi(s, a)$)

$$L(\theta) = \mathbb{E}_{s\sim P(\cdot)}\left[\left(V^\pi(s) - \hat{V}(s;\theta)\right)^2\right] \approx \mathbb{E}_{s\sim\mathcal{D}}\left[\left(V^\pi(s) - \hat{V}(s;\theta)\right)^2\right]$$

- Critical points: $\nabla_\theta L(\theta) = 0 \Leftrightarrow \sum_{s\in\mathcal{D}} \left(V^\pi(s) - \hat{V}(s;\theta)\right)\nabla_\theta \hat{V}(s;\theta) = 0$
  – Linear functions: analytical, global optimum solution. E.g., linear least squares methods (LSTD, LSPI)
- Gradient descent (GD) solution: batch (offline), SGD (incremental/online), mini-batch (hybrid)
  – SGD (stochastic gradient descent) *samples* the (full) gradient at each s or (s, a)
  – Least mean squares (LMS), aka Widrow-Hoff rule: SGD update

$$\theta_{i+1} = \theta_i + \alpha_i\left(V^\pi(s) - \hat{V}(s;\theta_i)\right)\nabla_{\theta_i}\hat{V}(s;\theta_i)$$

- Learned behaviors are often unstable/diverge when θ is a nonlinear (e.g., deep convolutional) function ☹
  – *Nonstationary* distribution: when θ is updated $\to$ policy/behavior & data distribution $P(\cdot)$ changed!
  – *Non-i.i.d.* training data distribution: samples are correlated (generated by interaction, in trajectory)
  – *Unstable* training due to correlations between $\hat{V}(s;\theta)$ and its regression target values (which depend on θ; see next slide).
- Deep Q-Network (DQN) provides a fix:
  1. Combined with *experience replay*: Store real-world experience in $\mathcal{D}$ + (i.i.d) sampling for SGD
  – Help remove correlations in the observation sequence & smooth over changes in $P(\cdot)$.
  2. Using a *fixed target network* ($\bar{\theta}$, updated slower) thereby reducing correlations with the target.

Learning Approximate Value Functions

2. Estimating Target Value

- No *true* value functions $V^{\pi}(s)/Q^{\pi}(s,a)$ are given as regression target.
  – use an *estimate* $\tilde{V}^{\pi}$ in place of target $V^{\pi}(s)$: $\theta_{i+1} = \theta_i + \alpha_i\big(\tilde{V}^{\pi}(s) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$

Learning Approximate Value Functions

2. Estimating Target Value

- No *true* value functions $V^{\pi}(s)/Q^{\pi}(s,a)$ are given as regression target.
  – use an *estimate* $\tilde{V}^{\pi}$ in place of target $V^{\pi}(s)$: $\theta_{i+1} = \theta_i + \alpha_i\big(\tilde{V}^{\pi}(s) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$
- MC: $\theta_{i+1} = \theta_i + \alpha_i\big(G(s) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$ (only updated at the *end* of each episode)

Learning Approximate Value Functions

2. Estimating Target Value

- No *true* value functions $V^\pi(s)/Q^\pi(s,a)$ are given as regression target.
  – use an *estimate* $\tilde{V}^\pi$ in place of target $V^\pi(s)$: $\theta_{i+1} = \theta_i + \alpha_i\big(\tilde{V}^\pi(s) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$
- MC: $\theta_{i+1} = \theta_i + \alpha_i\big(G(s) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$ (only updated at the *end* of each episode)
- TD(0): $\theta_{i+1} = \theta_i + \alpha_i\big(r + \gamma\hat{V}(s';\theta_i) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$ (updated at each step)

Learning Approximate Value Functions

2. Estimating Target Value

- No *true* value functions $V^\pi(s)/Q^\pi(s,a)$ are given as regression target.
  – use an *estimate* $\tilde{V}^\pi$ in place of target $V^\pi(s)$: $\theta_{i+1} = \theta_i + \alpha_i\big(\tilde{V}^\pi(s) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$
- MC: $\theta_{i+1} = \theta_i + \alpha_i\big(G(s) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$ (only updated at the *end* of each episode)
- TD(0): $\theta_{i+1} = \theta_i + \alpha_i\big(r + \gamma\hat{V}(s';\theta_i) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$ (updated at each step)
- Similarly, for control problems: use estimate $\tilde{Q}^\pi(s)$ for action-value function $Q^*(s,a)$

$$\text{SARSA (on-policy): } \theta_{i+1} = \theta_i + \alpha_i\big(r + \gamma\hat{Q}(s',a';\theta_i) - \hat{Q}(s,a;\theta_i)\big)\nabla_{\theta_i}\hat{Q}(s,a;\theta_i)$$

$$\text{Q-learning (off-policy): } \theta_{i+1} = \theta_i + \alpha_i\big(r + \gamma\max_{a'}\hat{Q}(s',a';\theta_i) - \hat{Q}(s,a;\theta_i)\big)\nabla_{\theta_i}\hat{Q}(s,a;\theta_i)$$

Learning Approximate Value Functions

2. Estimating Target Value

- No *true* value functions $V^\pi(s)/Q^\pi(s,a)$ are given as regression target.
  – use an *estimate* $\tilde{V}^\pi$ in place of target $V^\pi(s)$: $\theta_{i+1} = \theta_i + \alpha_i\big(\tilde{V}^\pi(s) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$
- MC: $\theta_{i+1} = \theta_i + \alpha_i\big(G(s) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$ (only updated at the *end* of each episode)
- TD(0): $\theta_{i+1} = \theta_i + \alpha_i\big(r + \gamma\hat{V}(s';\theta_i) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$ (updated at each step)
- Similarly, for control problems: use estimate $\tilde{Q}^\pi(s)$ for action-value function $Q^*(s,a)$

$$\text{SARSA (on-policy): } \theta_{i+1} = \theta_i + \alpha_i\big(r + \gamma\hat{Q}(s',a';\theta_i) - \hat{Q}(s,a;\theta_i)\big)\nabla_{\theta_i}\hat{Q}(s,a;\theta_i)$$

$$\text{Q-learning (off-policy): } \theta_{i+1} = \theta_i + \alpha_i\big(r + \gamma\max_{a'}\hat{Q}(s',a';\theta_i) - \hat{Q}(s,a;\theta_i)\big)\nabla_{\theta_i}\hat{Q}(s,a;\theta_i)$$

- Note that these regression targets depend on the network weights θ (causing correlations); this is in contrast with the targets used for "standard" supervised learning, which are fixed before learning begins.

Learning Approximate Value Functions

2. Estimating Target Value

- No *true* value functions $V^{\pi}(s)/Q^{\pi}(s,a)$ are given as regression target.
  – use an *estimate* $\tilde{V}^{\pi}$ in place of target $V^{\pi}(s)$: $\theta_{i+1} = \theta_i + \alpha_i \big(\tilde{V}^{\pi}(s) - \hat{V}(s;\theta_i)\big) \nabla_{\theta_i} \hat{V}(s;\theta_i)$
- MC: $\theta_{i+1} = \theta_i + \alpha_i \big(G(s) - \hat{V}(s;\theta_i)\big) \nabla_{\theta_i} \hat{V}(s;\theta_i)$ (only updated at the *end* of each episode)
- TD(0): $\theta_{i+1} = \theta_i + \alpha_i \big(r + \gamma \hat{V}(s';\theta_i) - \hat{V}(s;\theta_i)\big) \nabla_{\theta_i} \hat{V}(s;\theta_i)$ (updated at each step)
- Similarly, for control problems: use estimate $\tilde{Q}^{\pi}(s)$ for action-value function $Q^*(s,a)$

$$\text{SARSA (on-policy): } \theta_{i+1} = \theta_i + \alpha_i \big(r + \gamma \hat{Q}(s',a';\theta_i) - \hat{Q}(s,a;\theta_i)\big) \nabla_{\theta_i} \hat{Q}(s,a;\theta_i)$$

$$\text{Q-learning (off-policy): } \theta_{i+1} = \theta_i + \alpha_i \big(r + \gamma \max_{a'} \hat{Q}(s',a';\theta_i) - \hat{Q}(s,a;\theta_i)\big) \nabla_{\theta_i} \hat{Q}(s,a;\theta_i)$$

- Note that these regression targets depend on the network weights θ (causing correlations); this is in contrast with the targets used for "standard" supervised learning, which are fixed before learning begins.
- Deep Q-Network (DQN): $\theta_{i+1} = \theta_i + \alpha_i \big(r + \gamma \max_{a'} \hat{Q}(s',a';\bar{\theta}_i) - \hat{Q}(s,a;\theta_i)\big) \nabla_{\theta_i} \hat{Q}(s,a;\theta_i)$

Learning Approximate Value Functions

2. Estimating Target Value

- No *true* value functions $V^\pi(s)/Q^\pi(s,a)$ are given as regression target.
  – use an *estimate* $\tilde{V}^\pi$ in place of target $V^\pi(s)$: $\theta_{i+1} = \theta_i + \alpha_i\big(\tilde{V}^\pi(s) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$
- MC: $\theta_{i+1} = \theta_i + \alpha_i\big(G(s) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$ (only updated at the *end* of each episode)
- TD(0): $\theta_{i+1} = \theta_i + \alpha_i\big(r + \gamma\hat{V}(s';\theta_i) - \hat{V}(s;\theta_i)\big)\nabla_{\theta_i}\hat{V}(s;\theta_i)$ (updated at each step)
- Similarly, for control problems: use estimate $\tilde{Q}^\pi(s)$ for action-value function $Q^*(s,a)$

$$\text{SARSA (on-policy): } \theta_{i+1} = \theta_i + \alpha_i\big(r + \gamma\hat{Q}(s',a';\theta_i) - \hat{Q}(s,a;\theta_i)\big)\nabla_{\theta_i}\hat{Q}(s,a;\theta_i)$$

$$\text{Q-learning (off-policy): } \theta_{i+1} = \theta_i + \alpha_i\big(r + \gamma\max_{a'}\hat{Q}(s',a';\theta_i) - \hat{Q}(s,a;\theta_i)\big)\nabla_{\theta_i}\hat{Q}(s,a;\theta_i)$$

- Note that these regression targets depend on the network weights θ (causing correlations); this is in contrast with the targets used for "standard" supervised learning, which are fixed before learning begins.
- Deep Q-Network (DQN): $\theta_{i+1} = \theta_i + \alpha_i\big(r + \gamma\max_{a'}\hat{Q}(s',a';\bar{\theta}_i) - \hat{Q}(s,a;\theta_i)\big)\nabla_{\theta_i}\hat{Q}(s,a;\theta_i)$
- AlphaGo Zero: use MC target but train for *all* steps at the end of each episode (hence more labeled data).
  – Each return $G(s) \in \{0, \pm 1\}$ is from a trajectory with behavior policy *improved* by MCTS at each step.

Note: if s' is a *terminal state* then its (exact & approximated) value must be 0.