[San Jose State University Special AI Lecture Series VIII - RL/Recent Progress/App Development]
From Reinforcement Learning to Production - RL, Recent Breakthroughs & App Development

Sunghee Yun

Co-Founder & CTO @ Erudio Bio, Inc.
Co-Founder & CEO @ Erudio Bio Korea, Inc.
Leader of Silicon Valley Privacy-Preserving AI Forum (K-PAI)
CGO / Global Managing Partner @ LULUMEDIC
Global Leadership Initiative Fellow @ Salzburg Global Seminar
Visiting Professor & Advisory Professor @ Sogang Univ. & DGIST

About Speaker

- *Co-Founder & CTO @ Erudio Bio, Inc., San Jose & Novato, CA, USA* 2023 ∼
- *Co-Founder & CEO @ Erudio Bio Korea, Inc., Korea* 2025 ∼
- *Leader of Silicon Valley Privacy-Preserving AI Forum (K-PAI), CA, USA* 2024 ∼
- *CGO / Global Managing Partner @ LULUMEDIC, Seoul, Korea* 2025 ∼
- *KFAS-Salzburg Global Leadership Fellow @ Salzburg Global Seminar, Austria* 2024 ∼
- *Adjunct Professor, EE Department @ Sogang University, Seoul, Korea* 2020 ∼
- *Advisory Professor, EECS Department @ DGIST, Korea* 2020 ∼
- *AI-Korean Medicine Integration Initiative Task Force Member @ The Association of Korean Medicine, Seoul, Korea* 2025 ∼
- *Director of AI Semiconductor @ K-BioX, CA, USA* 2025 ∼
- Global Advisory Board Member @ Innovative Future Brain-Inspired Intelligence System Semiconductor of Sogang University, Korea 2020 ∼
- Technology Consultant @ Gerson Lehrman Gruop (GLG), NY, USA 2022 ∼
- Chief Business Development Officer @ WeStory.ai, Cupertino, CA, USA 2025 ∼
- Advisor @ CryptoLab, Inc., Seoul, Korea 2025 ∼

- Co-Founder & CTO / Head of Global R&D / Chief Applied Scientist / Senior Fellow @ Gauss Labs, Inc., Palo Alto, CA, USA 2020 ∼ 2023
- Senior Applied Scientist @ Amazon.com, Inc., Vancouver, BC, Canada 2017 ∼ 2020
- Principal Engineer @ Software R&D Center, Samsung Electronics 2016 ∼ 2017
- Principal Engineer @ Strategic Marketing & Sales, Memory Business 2015 ∼ 2016
- Principal Engineer @ DT Team, DRAM Development, Samsung 2012 ∼ 2015
- Senior Engineer @ CAE Team, Memory Business, Samsung, Korea 2005 ∼ 2012
- PhD - Electrical Engineering @ Stanford University, CA, USA 2001 ∼ 2004
- Development Engineer @ Voyan, Santa Clara, CA, USA 2000 ∼ 2001
- MS - Electrical Engineering @ Stanford University, CA, USA 1998 ∼ 1999
- BS - Electrical & Computer Engineering @ Seoul National University 1994 ∼ 1998

Highlight of Career Journey

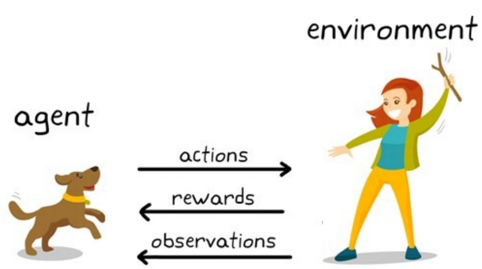
- BS in Electrical Engineering (EE) @ Seoul National University
- MS & PhD in Electronics Engineering (EE) @ Stanford University
  - *Convex Optimization - Theory, Algorithms & Software*
  - advisor - *Prof. Stephen P. Boyd*
- Principal Engineer @ Samsung Semiconductor, Inc.
  - *AI & Convex Optimization*
  - collaboration with *DRAM/NAND Design/Manufacturing/Test Teams*
- Senior Applied Scientist @ Amazon.com, Inc.
  - *e-Commerce AIs* - anomaly detection, deep RL, and recommender system
  - *Jeff Bezos's project - drove $200M* in sales via Amazon Mobile Shopping App
- *Co-Founder & CTO / Global R&D Head & Chief Applied Scientist* @ Gauss Labs, Inc.
- *Co-Founder & CTO* @ Erudio Bio, Inc.
- *Co-Founder & CEO* @ Erudio Bio Korea, Inc.

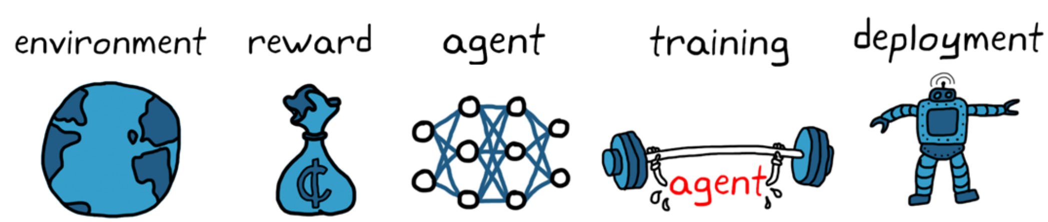
Unpacking AI

Reinforcement Learning

Reinforcement learning (RL)

- machine learning where agent learns how to take actions to achieve goal
  - by maximizing cumulative *reward*
  - while interacting with environment
- learning from interaction - foundational idea underlying all learning & intelligence
- differs from supervised learning
  - labeled input and output pairs *not* presented
  - sub-optimal actions need *not* be explicitly corrected
- focus is finding balance between exploration & exploitation





Why Deep RL?

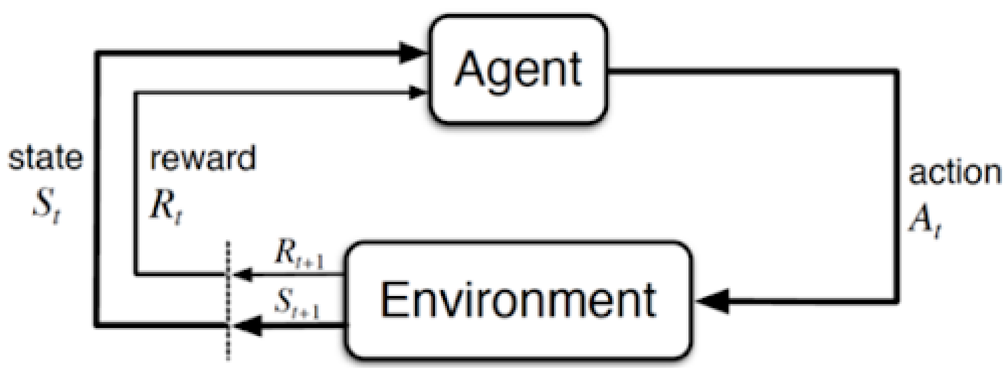
- Koray Kavukcuoglu (director of research at Deepmind) says

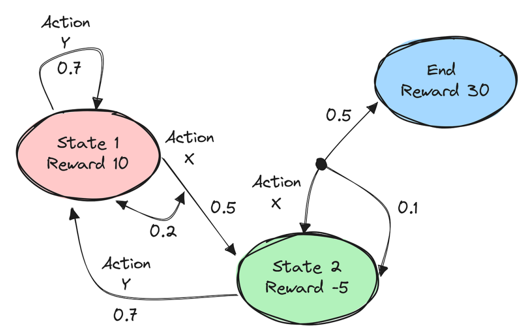
> If one of the goals we work for here is AI, then it is at the core of that. RL is a very general framework for learning sequential decision making tasks. And DL, on the other hand, is (of course) the best set of algorithms we have to learn representations. And combinations of these two different models is the best answer so far we have in terms of learning very good state representations of very challenging tasks that are not just for solving toy domains but actually to solve challenging real world problems.

MDP

Markov decision process (MDP)

- classical formulation of sequential decision making
  - actions influence not just immediate rewards, but also subsequent states, hence, involving delayed reward
  - need to trade-off immediate and delayed reward
- elements - *states*, *actions*, *reward*, and *return*
- agent interacts with environment
  - agent makes decision as to which action to take with knowledge of state it's in
  - action changes (state of) environment
  - agent receives reward

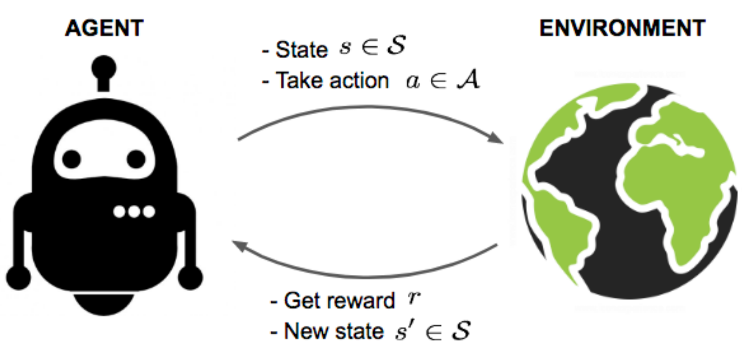


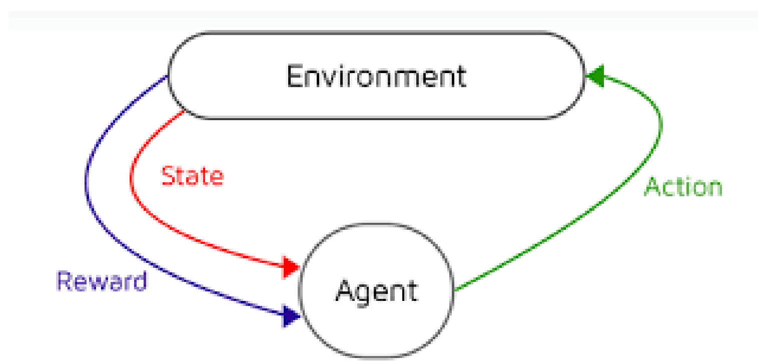


MDP & Markov property

- agent in *state* S_t takes *action* A_t at t
  - receives *reward* R_{t+1} (from environment)
  - environment transitions to state S_{t+1}
- sequence of random variables - $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, \ldots$
- *Markov property* - $S_{t+1}, R_{t+1} | S_t, A_t, R_t, S_{t-1}, A_{t-1}, R_{t-1}, \ldots = S_{t+1}, R_{t+1} | S_t, A_t$
  - formally expressed (using PDF)

$$p\left(S_{t+1}, R_{t+1} \middle| S_t, A_t, R_t, S_{t-1}, A_{t-1}, R_{t-1}, \ldots\right) = p\left(S_{t+1}, R_{t+1} \middle| S_t, A_t\right)$$



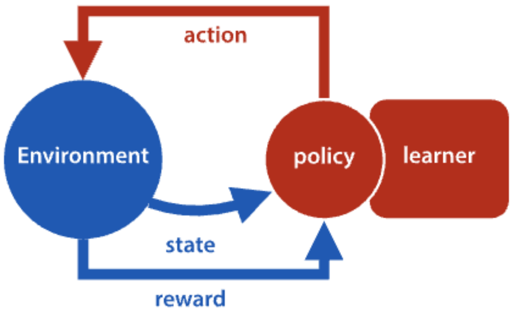


Policy & return

- *policy* - conditional probability of A_t given S_t

$$\pi(A|S) = p(A_t|S_t),$$

- *return* (at t) - $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$
- $\gamma \in [0, 1]$ - *discount factor*
  - if $\gamma = 0$, myopic
  - if $\gamma = 1$, truly far-sighted
  - if $\gamma \in (0, 1)$, considers near-future rewards more importantly than those in far future



State value function & action value function

- *state value function* (sometimes referred to simply as *value function*)

$$v_\pi(s) = \mathop{\mathbf{E}}_{\pi,p} \{G_t | \, S_t = s\} = \mathop{\mathbf{E}}_{\pi,p} \left\{ \left. \sum_{k=0}^{\infty} \gamma^k R_{t+k} \right| S_t = s \right\}$$

  – function of state - expected return agent will get from s when following π

- *action value function* (sometimes referred to simply as *action function*)

$$q_\pi(s,a) = \mathop{\mathbf{E}}_{\pi,p} \{G_t | \, S_t = s, A_t = a\} = \mathop{\mathbf{E}}_{\pi,p} \left\{ \left. \sum_{k=0}^{\infty} \gamma^k R_{t+k} \right| S_t = s, A_t = a \right\}$$

  – function of state & action - expected return agent will get from s when agent takes a

- (most) RL algorithms (try to) maximize either of these functions - not maximizing immediate reward, but long-term return

Bellman

- Richard E. Bellman
  - introduced dynamic programming (DP) in 1953
  - proposed *Bellman equation* as necessary condition for optimality associated with DP

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right] \\
&\overset{?}{=} \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \Big[r + \gamma v_\pi(s')\Big], \quad \forall s \in \mathcal{S}, \qquad (3.12)
\end{aligned}
$$

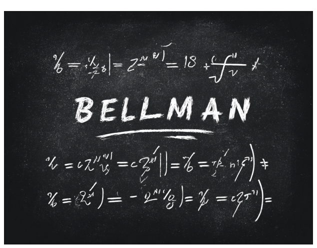
Bellman equations

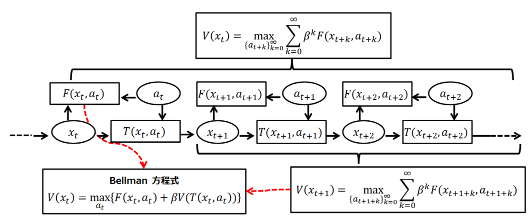
- *Bellman equation for state value function*

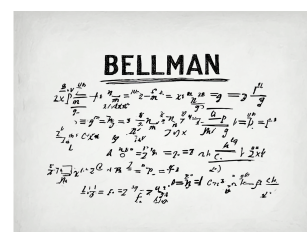
$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left(r + \gamma v_\pi(s')\right) \quad (1)$$

- *Bellman equation for action value function*

$$\begin{aligned} q_\pi(s,a) &= \sum_{s',r} p(s',r|s,a) \left(r + \gamma v_\pi(s')\right) \\ &= \sum_{s',r} p(s',r|s,a) \left(r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')\right) \end{aligned} \quad (2)$$







Bellman equation derviation - state value function

- Markov property implies
  - value functions only depend on current state & action taken
  - function value closely related to function values of next states
- these facts cleverly used to derive Bellman equations

$$
\begin{aligned}
v_\pi(s) &= \mathop{\mathbf{E}}_{\pi,p} \{ G_t | \, S_t = s \} \\
&= \mathop{\mathbf{E}}_{A_t|S_t=s} \mathop{\mathbf{E}}_{\pi,p} \{ G_t | \, S_t = s, A_t \} \\
&= \sum_a p(A_t = a | S_t = s) \mathop{\mathbf{E}}_{\pi,p} \{ G_t | \, S_t = s, A_t = a \} \\
&= \sum_a \pi(a|s) \mathop{\mathbf{E}}_{\pi,p} \{ G_t | \, S_t = s, A_t = a \} \\
&= \sum_a \pi(a|s) q_\pi(s,a) \qquad (3)
\end{aligned}
$$

Bellman equation derviation - action value function

$$
\begin{aligned}
q_\pi(s,a) &= \mathop{\mathbf{E}}_{\pi,p} \left\{ G_t |\, S_t = s, A_t = a \right\} \\
&= \mathop{\mathbf{E}}_{S_{t+1},R_{t+1}|S_t=s,A_t=a} \mathop{\mathbf{E}}_{\pi,p} \left\{ G_t |\, S_t = s, A_t = a, S_{t+1}, R_{t+1} \right\} \\
&= \mathop{\mathbf{E}}_{S_{t+1},R_{t+1}|S_t=s,A_t=a} \mathop{\mathbf{E}}_{\pi,p} \left\{ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle|\, S_t = s, A_t = a, S_{t+1}, R_{t+1} \right\} \\
&= \mathop{\mathbf{E}}_{S_{t+1},R_{t+1}|S_t=s,A_t=a} \mathop{\mathbf{E}}_{\pi,p} \left\{ R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \middle|\, S_t = s, A_t = a, S_{t+1}, R_{t+1} \right\} \\
&= \sum_{s',r} p_{S_{t+1},R_{t+1}|S_t,A_t}(s',r|s,a) \\
&\qquad \mathop{\mathbf{E}}_{\pi,p} \left\{ R_{t+1} + \gamma G_{t+1} |\, S_t = s, A_t = a, S_{t+1} = s', R_{t+1} = r \right\}
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{s',r} p_{S_{t+1},R_{t+1}|S_t,A_t}(s',r|s,a) \\
&\qquad \left(r + \gamma \mathop{\mathbf{E}}_{\pi,p} \left\{ G_{t+1} | \, S_t = s, A_t = a, S_{t+1} = s', R_{t+1} = r \right\} \right) \\
&= \sum_{s',r} p_{S_{t+1},R_{t+1}|S_t,A_t}(s',r|s,a) \left(r + \gamma \mathop{\mathbf{E}}_{\pi,p} \left\{ G_{t+1} | \, S_{t+1} = s' \right\} \right) \\
&= \sum_{s',r} p_{S_{t+1},R_{t+1}|S_t,A_t}(s',r|s,a) \left(r + \gamma v_\pi(s') \right) \qquad (4)
\end{aligned}
$$

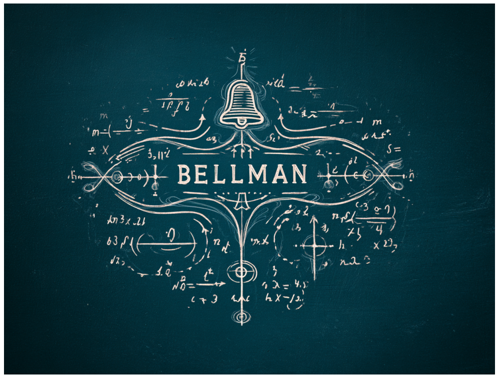
Optimal functions

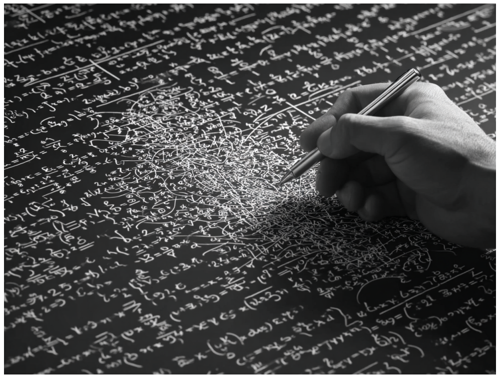
- define *optimal state-value function* as that of optimal policy π_*

$$v_*(s) = v_{\pi_*}(s) = \max_{\pi \in \Pi} v_\pi(s) \tag{5}$$

- (similarly) define *optimal action-value function* as that of π_*

$$q_*(s,a) = q_{\pi_*}(s,a) = \max_{\pi \in \Pi} q_\pi(s,a) \tag{6}$$





Bellman optimality equations

(5) & (6) with (3) & (4) imply

- *Bellman optimality equation for state value function*

$$v_*(s) = v_{\pi*}(a) = \max_{a \in \mathcal{A}} q_{\pi*}(s,a) = \max_{a \in \mathcal{A}} \sum_{s',r} p(s',r|s,a) \left(r + \gamma v_\pi(s')\right) \quad (7)$$

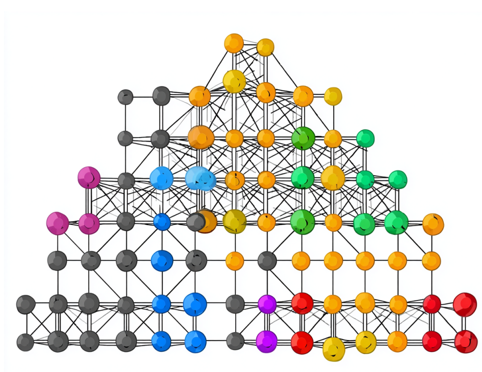
- *Bellman optimality equation for action value function*

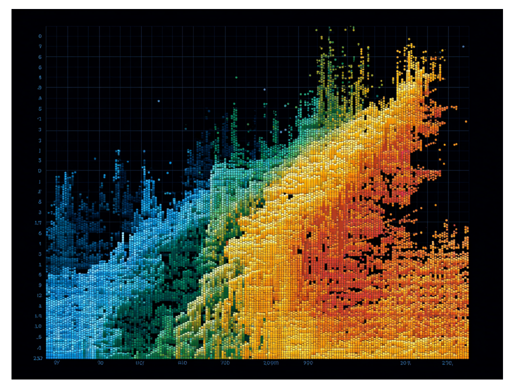
$$\begin{aligned} q_*(s,a) &= q_{\pi*}(s,a) = \sum_{s',r} p(s',r|s,a) \left(r + \gamma v_{\pi*}(s')\right) \\ &= \sum_{s',r} p(s',r|s,a) \left(r + \gamma \max_{a' \in \mathcal{A}} q_{\pi*}(s',a')\right) \quad (8) \end{aligned}$$

Dynamic Programming

Dynamic programming (DP)

- collection of algorithms to *compute optimal policies given perfect model of environment as MDP*
- provide *essential foundation for understanding of RL methods*
- all RL algorithms can be viewed as attempts to achieve much the same effect as DP
  - only with *less computation and without assuming perfect model of environment*
- key idea of RL in general
  - use of *value functions* to organize and structure search for good policies

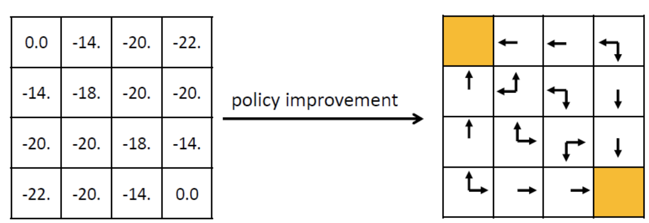




Policy evaluation (prediction)

- *policy evaluation* (in DP literature)
  - compute state-value function v_π for arbitrary policy π
  - also referred to as *prediction problem*
- existence and uniqueness of v_π guaranteed as long as either
  - $\gamma < 1$
  - eventual termination is guaranteed from all states under policy π
- policy evaluation algorithm uses fact that all state value functions satisfy Bellman equation (note resemblance to 1) - algorithm described in Table 1

$$v_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left(r + \gamma v_k(s')\right)$$



Algorithm - iterative policy evaluation

Inputs: π, MDP
Algorithm parameters: $\theta > 0$ (small threshold determining accuracy of estimation)

Initialize $V(s) \in \mathbf{R}$ for all $s \in \mathcal{S}$ except that $V(\mathrm{terminal}) = 0$

Loop:
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left(r + \gamma V(s')\right)$
        $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
until $\Delta < \theta$

Table 1: Iterative Policy Evaluation for estimating $V \sim v_\pi$

Policy iteration

- iterative process of improving policy to maximize value functions

- algorithm described in Table 2

Algorithm - policy iteration

Inputs: MDP
Algorithm parameters: $\theta > 0$ (small threshold determining accuracy of estimation)
1. Initialization
    $V(s) \in \mathbf{R}$ and $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$
2. Policy Evaluation
    Loop:
        $\Delta \leftarrow 0$
        For each $s \in \mathcal{S}$:
            $v \leftarrow V(s)$
            $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left(r + \gamma V(s')\right)$
            $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
    until $\Delta < \theta$
3. Policy Improvement
$u \leftarrow \texttt{true}$
    For each $s \in \mathcal{S}$
        $b \leftarrow \pi(s)$
        $\pi(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left(r + \gamma v_\pi(s')\right)$
        If $b \neq \pi(s)$, then $t \leftarrow \texttt{false}$
    If u, then stop and return $V \sim v_*$ and $\pi \sim \pi_*$; else go to 2

Table 2: Policy Iteration (using iterative policy evaluation) for estimating $\pi \sim \pi_*$

Value iteration

- drawback to policy iteration
  - each iteration involves policy evaluation
- policy evaluation step can be truncated without losing convergence guarantees
- *value iteration*
  - policy evaluation is stopped after just one sweep by turning Bellman optimality equation (7) into update rule
  - can be written as simple update operation combining policy improvement and truncated policy evaluation steps

$$v_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s',r} p(s', r|s, a) \left(r + \gamma v_k(s')\right)$$

- (in-place version of) algorithm described in Table 3

Algorithm - value iteration

Inputs: MDP
Algorithm parameters: $\theta > 0$ (small threshold determining accuracy of estimation)

Initialize $V(s) \in \mathbf{R}$ for all $s \in \mathcal{S}$ except that $V(\mathrm{terminal}) = 0$

Loop:
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r|s, a) \left(r + \gamma V(s')\right)$
    $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
until $\Delta < \theta$

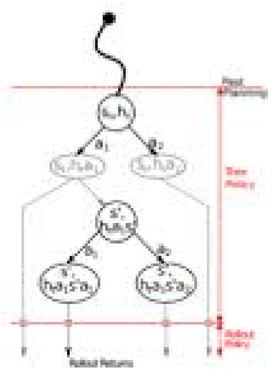
Output: deterministic policy π such that
  $\pi(s) = \mathrm{argmax}_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r|s, a) \left(r + \gamma V(s')\right)$

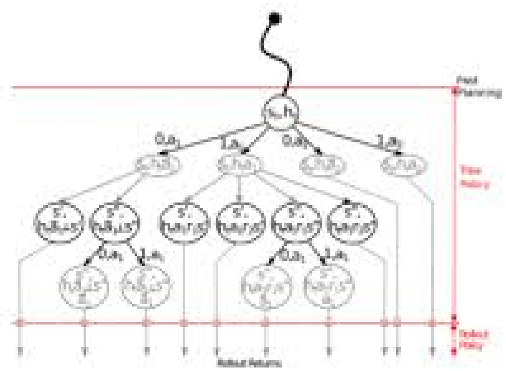
Table 3: Value Iteration for estimating $\pi \sim \pi_*$

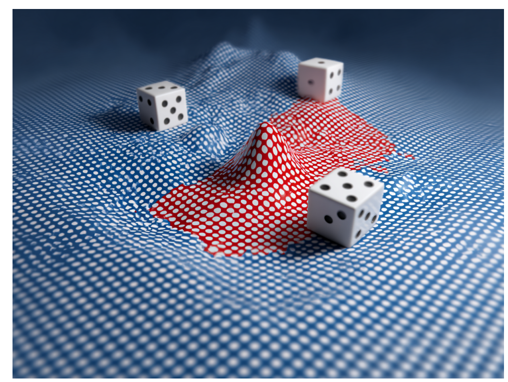
Monte Carlo Methods

Monte Carlo methods

- do not assume complete knowledge of environment
- require only experience sample sequences of states, actions & rewards
  - from *actual or simulated interaction with environment*
- require no prior knowledge of environment's dynamics
  - *not complete probability distributions* required for DP
  - yet can still attain optimal behavior
- simulation can be used







Monte Carlo prediction

- (simply) average returns observed after visits to each state
- Monte Carlo (MC) prediction methods - very similar but slightly different theoretical properties
  - *first-visit MC method* - most widely studied, dating back to 1940s
  - *every-visit MC method* - extends more naturally to function approximation and eligibility traces
- first-visit MC prediction algorithm described in Table 4

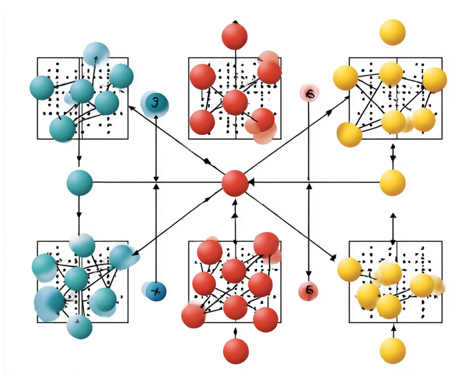
Algorithm - first-visit MC prediction

Inputs: π
Initialize:
$V(s) \in \mathbf{R}$ for all $s \in \mathcal{S}$
$R(s) \leftarrow \texttt{list}()$ for all $s \in \mathcal{S}$
Loop:
Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
Loop for each step of episode, $t + T - 1, T - 2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
If $S_t \not\in \{S_0, S_1, \ldots, S_{t-1}\}$:
$R(S_t).\texttt{append}(G)$
$V(S_t) \leftarrow R(S_t).\texttt{average}()$
Until a certain criterion is satisfied

Table 4: First-visit MC prediction for estimating $V \sim v_\pi$

Monte Carlo control

- proceed according to same pattern as DP, *i.e.*, according to idea of generalized policy iteration (GPI)
- maintain both approximate policy & approximate value functions
  - value functions repeatedly altered to more closely approximate value function for current policy
  - policy repeatedly improved with respect to current value function
- complete simple algorithm, called *Monte Carlo with Exploring Starts (ES)* described in Table 5



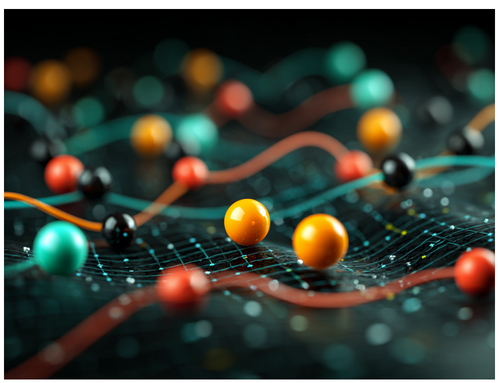
Algorithm - MC ES

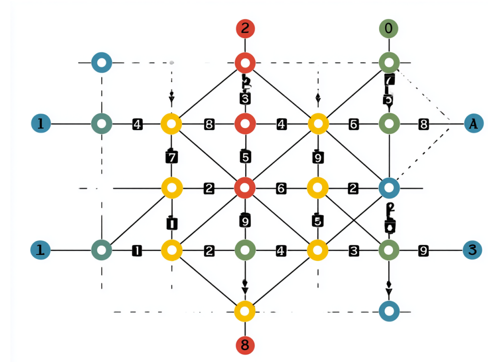
```
Initialize:
    π(s) ∈ 𝒜(s) for all s ∈ 𝒮
    Q(s, a) ∈ R for all s ∈ 𝒮 and a ∈ 𝒜(s)
    R(s, a) ← list() for all s ∈ 𝒮 and a ∈ 𝒜(s)

Loop:
    Choose S_0 ∈ 𝒮, A_0 ∈ 𝒜(S_0) randomly such that all pairs have probability > 0
    Generate an episode from S_0, A_0 following π: S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{T-1}, A_{T-1}, R_T
    G ← 0
    Loop for each step of episode, t + T - 1, T - 2, ..., 0:
        G ← γG + R_{t+1}
        If S_t ∉ {S_0, S_1, ..., S_{t-1}}:
            R(S_t, A_t).append(G)
            Q(S_t, A_t) ← R(S_t, A_t).average()
            π(S_t) ← argmax_{a∈𝒜(S_t)} Q(S_t, a)
Until a certain criterion is satisfied
```

Table 5: MC ES for estimating $\pi \sim \pi_*$

Monte Carlo control without exploring starts

- want to avoid unlikely assumption of exploring starts
- only general way to ensure that all actions are selected infinitely often is for agent to continue to select them
- two approaches to ensure this
  - on-policy methods - attempt to evaluate or improve policy used to make decisions
  - off-policy methods - evaluate or improve policy different from used to generate data
- on-policy first-visit MC control using ϵ-greedy, not using unrealistic assumption of exploring starts, described in Table 6





Algorithm - on-policy first-visit MC control

Algorithm parameters: small $\epsilon > 0$

Initialize:
- $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$
- $Q(s, a) \in \mathbf{R}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
- $R(s, a) \leftarrow \texttt{list}()$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Loop:
- Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
- Generate an episode from S_0, A_0 following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
- $G \leftarrow 0$
- Loop for each step of episode, $t + T - 1, T - 2, \ldots, 0$:
  - $G \leftarrow \gamma G + R_{t+1}$
  - If $S_t \not\in \{S_0, S_1, \ldots, S_{t-1}\}$:
    - $R(S_t, A_t).\texttt{append}(G)$
    - $Q(S_t, A_t) \leftarrow R(S_t, A_t).\texttt{average}()$
    - $A^* \leftarrow \mathrm{argmax}_{a \in \mathcal{A}(S_t)}$
    - For all $a \in \mathcal{A}(S_t)$

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \epsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

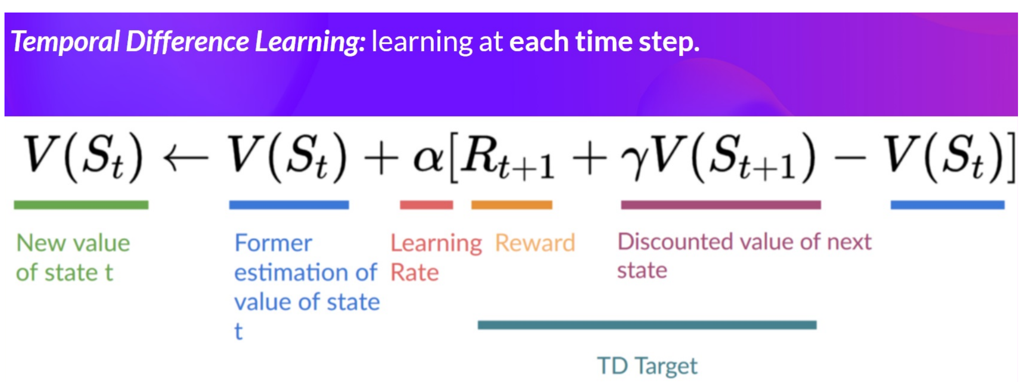
Until a certain criterion is satisfied

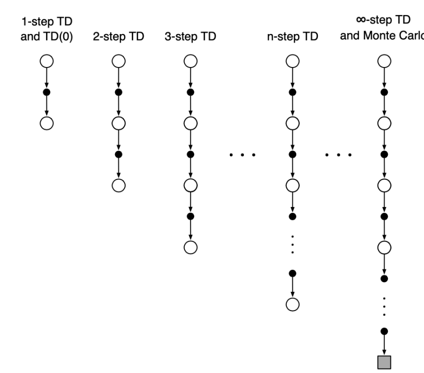
Table 6: On-policy first-visit MC control (for ϵ-soft policies) for estimating $\pi \sim \pi_*$

Temporal-difference Learning

Temporal-difference (TD) learning

- combination of MC ideas & DP ideas
  - like MC, learn directly from raw experience without model of environment's dynamics
  - like DP, update estimates based in part on other learned estimates, without waiting for final outcome - *they bootstrap*
- relationship between TD, DP & MC methods - recurring theme in theory of RL
- will start focusing on policy evaluation or prediction problem, $i.e.$, estimating v_π
- control problem (to find optimal policy)
  - DP, TD & MC methods all use some variation of generalized policy iteration (GPI)





TD prediction

- both TD & MC use experience to solve prediction problem
- simple every-visit MC method suitable for nonstationary environments

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) = (1-\alpha)V(S_t) + \alpha G_t$$

- TD methods wait only until next time step
  - at $t+1$, form target and make update using reward R_{t+1} & estimate $V(S_{t+1})$
- TD(0) - one-step TD - simplest TD method

$$\begin{aligned} V(S_t) \quad &\leftarrow \quad V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \\ &= (1-\alpha)V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1})) \end{aligned} \tag{9}$$

  - TD(0) is special case of TD(λ) & n-step TD methods
- TD(0) described in Table 7 in procedural form

Algorithm - TD(0) for estimating v_π

Inputs: the policy π to be evaluated
Algorithm parameters: step size $\alpha \in (0, 1]$

Initialize:
    $V(s) \in$ **R** for all $s \in \mathcal{S}$ except that $V(\mathrm{terminal}) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow (1 - \alpha)V(S) + \alpha(R + \gamma V(S'))$
        $S \leftarrow S'$
    until S is terminal
Until a certain criterion is satisfied

Table 7: TD(0) for estimating v_π.

TD error

- *TD error* - quantity in brackets in TD(0) update

$$\delta_t := R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t) \tag{10}$$

  - difference between estimated value of S_t & better estimate $R_{t+1} + \gamma V(S_{t+1})$
  - arise in various forms throughout RL

- define *modified TD error*

$$\delta'_t := R_{t+1} + \gamma V_{t+1}(S_{t+1}) - V_t(S_t) \tag{11}$$

Monte Carlo error

- MC error
  - difference between return along path from t to terminal state & state-value function

$$G_t - V_t(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta'_k = \sum_{k=0}^{T-t-1} \gamma^k \delta'_{k+t} \tag{12}$$

  - can be expressed as sum of discounted (modified) one-step TD errors.
- assuming that every V_t does not change during episode
  - δ_t coincides with δ'_t
  - hence, (12) becomes

$$G_t - V(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k = \sum_{k=0}^{T-t-1} \gamma^k \delta_{k+t}. \tag{13}$$

MC error - derivation

- MC error

$$
\begin{aligned}
G_t - V_t(S_t) = R_{t+1} + \gamma G_{t+1} - V_t(S_t) & \\
= \quad & R_{t+1} + \gamma \left(G_{t+1} - V_{t+1}(S_{t+1}) + V_{t+1}(S_{t+1}) \right) - V_t(S_t) \\
= \quad & R_{t+1} + \gamma V_{t+1}(S_{t+1}) - V_t(S_t) + \gamma \left(G_{t+1} - V_{t+1}(S_{t+1}) \right) \\
= \quad & \delta'_t + \gamma \left(G_{t+1} - V_{t+1}(S_{t+1}) \right) \\
= \quad & \delta'_t + \gamma \delta'_{t+1} + \gamma^2 \left(G_{t+2} - V_{t+2}(S_{t+2}) \right) \\
= \quad & \delta'_t + \gamma \delta'_{t+1} + \gamma^2 \delta'_{t+2} + \cdots + \gamma^{T-t-2} \delta'_{T-2} + \gamma^{T-t-1} \left(G_{T-1} - V_{T-1}(S_{T-1}) \right) \\
= \quad & \delta'_t + \gamma \delta'_{t+1} + \gamma^2 \delta'_{t+2} + \cdots + \gamma^{T-t-2} \delta'_{T-2} + \gamma^{T-t-1} \left(R_T + \gamma V_T(S_T) - V_{T-1}(S_{T-1}) \right) \\
= \quad & \delta'_t + \gamma \delta'_{t+1} + \gamma^2 \delta'_{t+2} + \cdots + \gamma^{T-t-2} \delta'_{T-2} + \gamma^{T-t-1} \delta'_{T-1} \\
= \quad & \sum_{k=t}^{T-1} \gamma^{k-t} \delta'_k = \sum_{k=0}^{T-t-1} \gamma^k \delta'_{k+t}
\end{aligned}
$$

where fact that state-value function for terminal state, $V_{T-1}(S_T)$, is 0 is used

Sarsa - on-policy TD Control

- (as in all on-policy methods)
  - continually estimate q_π for behavior policy π
  - (at the same time) change π toward greediness with respect to q_π
- convergence properties depend on nature of policy's dependence on Q
  - examples of policies - ϵ-greedy or ϵ-soft
- converges with probability 1 to an optimal policy & optimal action-value function as long as
  - all state–action pairs are visited infinite number of times
  - policy converges in the limit to greedy policy
- algorithm is described in Table 8

Algorithm - sarsa for estimating $Q \sim q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$ and small $\epsilon > 0$

Initialize:

    $Q(s, a) \in$ **R** for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ except $Q(\mathrm{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q ($e.g.$, ϵ-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q ($e.g.$, ϵ-greedy)
        $Q(S, A) \leftarrow (1 - \alpha)Q(S, A) + \alpha(R + \gamma Q(S', A'))$
        $S \leftarrow S'$, $A \leftarrow A'$,
    until S is terminal
Until a certain criterion is satisfied

Table 8: Sarsa (on-policy TD control) for estimating $Q \sim q_*$

Q-learning - off-policy TD control

- development of off-policy TD control algorithm known as Q-learning (Watkins, 1989) - one of early breakthroughs in RL
- update defined by

$$\begin{aligned} Q(S_t, A_t) \quad \leftarrow \quad & Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right) \\ & = (1-\alpha) Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) \right) \end{aligned}$$

- learned action-value function Q directly approximates optimal action-value function q_*, independent of policy being followed
  - dramatically simplifies analysis of algorithm & enabled early convergence proofs
- Q has been shown to converge with probability 1 to q_*
- algorithm described in Table 9

Algorithm - Q-learning for estimating $\pi \sim \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$ and small $\epsilon > 0$

Initialize:
  $Q(s, a) \in$ **R** for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ except $Q(\mathrm{terminal}, \cdot) = 0$

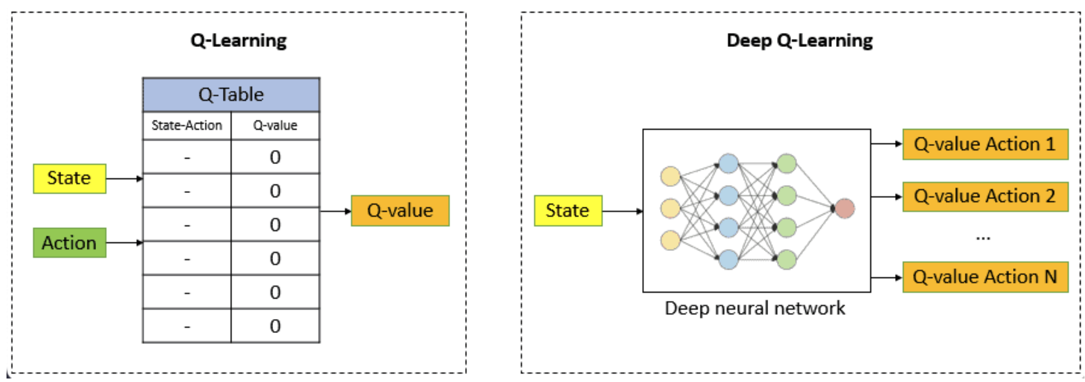
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q ($e.g.$, ϵ-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow (1 - \alpha)Q(S, A) + \alpha(R + \gamma \max_{a \in \mathcal{A}(S')} Q(S', a))$
    $S \leftarrow S'$
  until S is terminal
Until a certain criterion is satisfied

Table 9: Q-learning (off-policy TD control) for estimating $\pi \sim \pi_*$

Modern Reinforcement Learning

Deep Q-learning revolution

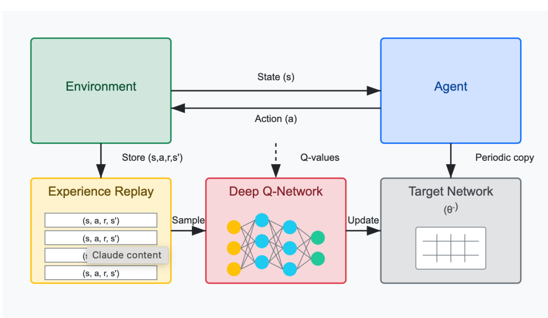
- problem with classical Q-learning
  - limited to small, discrete state spaces
  - Q-table becomes intractable for complex environments
  - cannot handle high-dimensional inputs, $e.g.$, images, continuous states
- deep Q-networks (DQN)
  - replace Q-table with deep neural network (DNN)
  - DNN approximates action-value function $Q(s,a)$
  - handle raw pixel inputs, continuous states
  - enables RL in complex environments, $e.g.$, Atari games, robotics

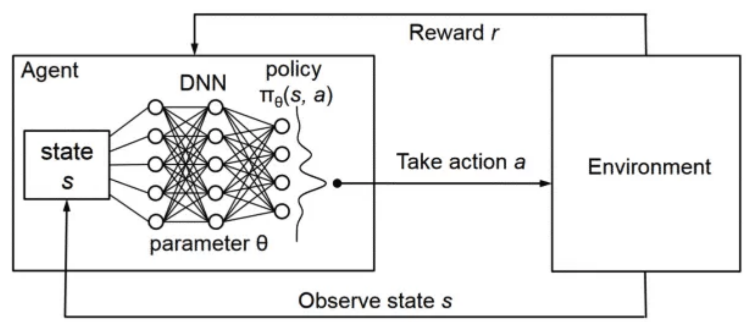


DQN architecture & key innovations

- experience replay
  - store transitions (s, a, r, s') in replay buffer & sample mini-batches for training
  - break correlation between consecutive samples to improve data efficiency and stability
- target network
  - separate target network for computing TD targets being updated periodically
  - reduce correlation between Q-values & targets to improve training stability
- DQN loss function

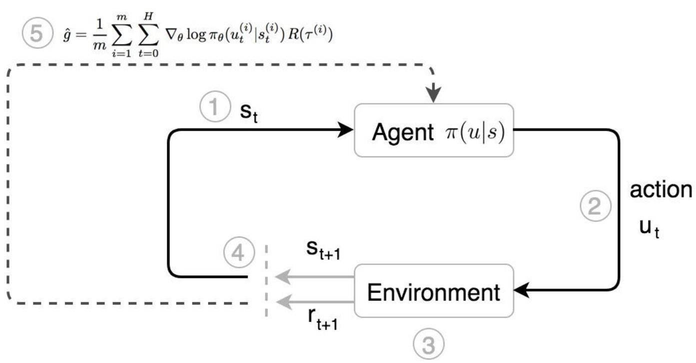
$$L(\theta) = \mathbf{E}((r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2$$

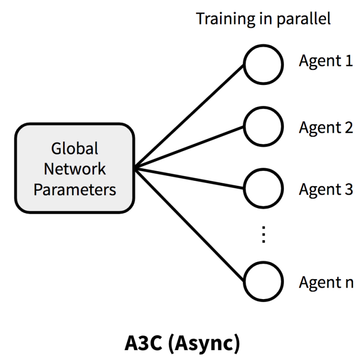


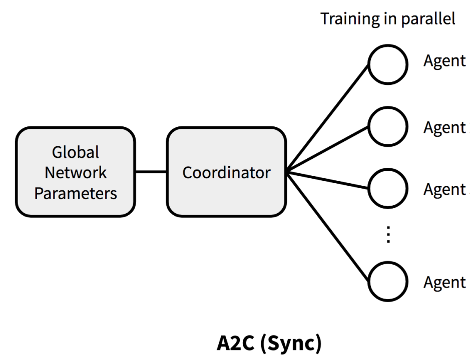


Policy gradient methods

- limitations of value-based approaches
  - indirect policy optimization
  - difficulty with continuous action spaces
  - may not find stochastic optimal policies
- policy gradient methods
  - direct policy optimization & natural handling of continuous actions
  - can learn stochastic policies & better convergence properties (in some cases)







Policy gradient algorithm

- merit cuntion - $J(\theta) = \mathbf{E}(V(S_0)|\pi_\theta) = \mathbf{E}\left(\sum_{t=0}^{\infty} \gamma^t R_t \middle| \pi_\theta\right)$
- maximization problem formulation

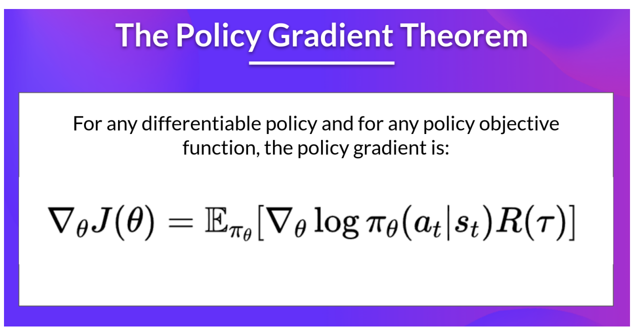
$$
\begin{array}{ll}
\text{maximize} & J(\theta) \\
\text{subject to} & \theta \in \Theta
\end{array}
$$

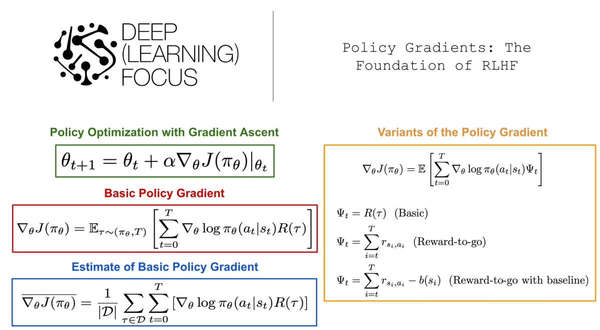
- REINFORCE algorithm

$$\theta^{k+1} = \theta^k + \alpha^k \nabla J(\theta^k)$$

where

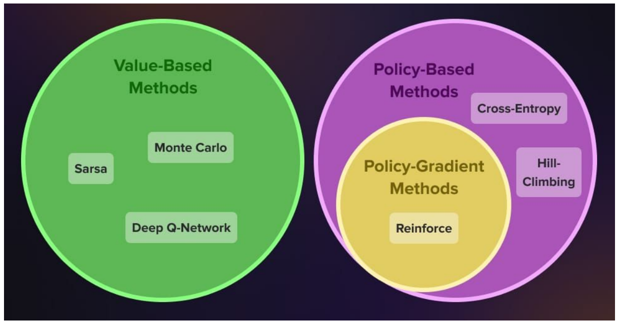
$$\nabla_\theta J(\theta) = \mathbf{E}(\nabla_\theta \log \pi(a|s;\theta) Q^\pi(s,a))$$

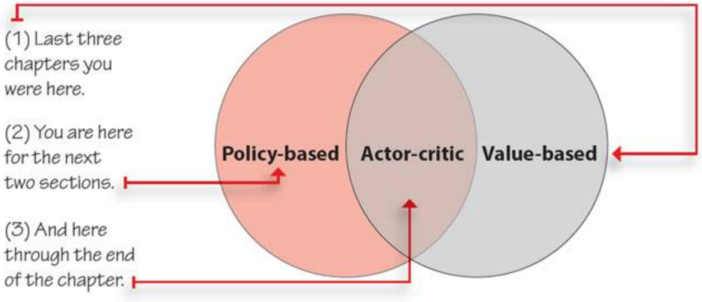




Q-learning vs policy gradients

- Q-learning
  - does *not* always work
  - usually *more sample-efficient* (when it works)
  - *challenge - exploration*
  - *no guarantee* for convergence
- policy gradients
  - *very general*, but suffers from *high variance*
  - requires *lots of samples*
  - converges to local minima of $J(\theta)$
  - *challenge - sample-efficiency*





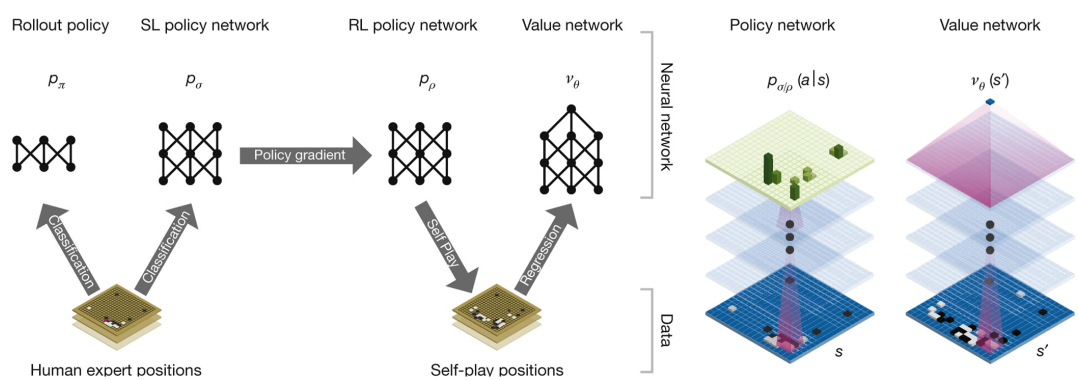
AlphaGo & AlphaGo Zero Technologies

AlphaGo

- Go board marked with 19×19 grid - $(19 \times 19)! = 361! \approx 1.44 \times 10^{768}$
  1437923258884890654832362511499863354754907538644755876127282765299227795534389618856841908003141196
  0714137944348905859683839682333043216077138088370565578796691924861827097800358990211005794501073330
  5079262777172275041226808677528136885057526541812043502150623466302643442673632627092764643302557772
  2695595343233942204301825548143785112222186834487969871267194205609533306413935710635197200721473378
  7338269803085351043174203653673779887217565513450041291061650506154496265581102824241428406627054585
  5623101563752892899924857388316647687165212001536218913733713768261861456295440900774337589490771443
  9917299937133680728459000034496420337066440853337001284286412654394495050773954560000000000000000000
  000000000000000000000000000000000000000000000000000000000000000000000
- deep reinforcement learning with Monte Carlo tree search
  - trained on thousands of years of Go game history
  - AlphaGo Zero learns by playing against itself
- development experience, insight, knowledge, know-how transferred to AlphaFold

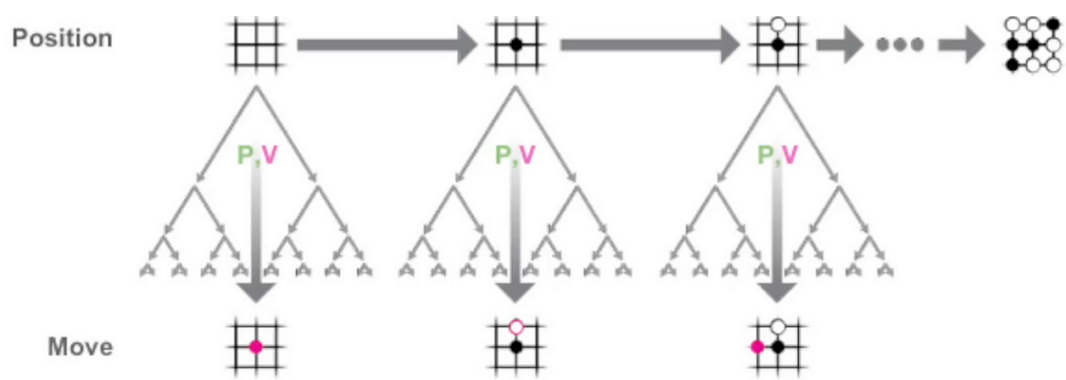
AlphaGo - hybrid approach - 2016

- components
  - policy network - predicts human expert moves
  - value network - evaluates board positions
  - Monte Carlo tree search (MCTS) - explores game tree
  - rollout policy - fast playouts for MCTS
- training process
  - supervised learning - train policy network on human games
  - RL - improve policy through self-play
  - regression - train value network on self-play positions



AlphaGo Zero - pure RL revolution - 2017

- breakthrough - no human knowledge
  - learns from scratch through self-play, no human game data or handcrafted features
  - much stronger than original AlphaGo
- simplified architecture
  - single neural network with two heads - policy head $\pi(a|s)$ & value head $v(s)$
- key innovations
  - residual NN - enable very deep networks
  - MCTS with NN - perfect integration
  - self-play curriculum - gradually increasing difficulty



Modern RL Applications & Industry Examples

Autonomous systems

- Waymo - Google
  - RL for trajectory planning and decision making
  - Simulation-based training with millions of scenarios
  - Integration with traditional planning algorithms
- Tesla Autopilot
  - RL for lane changes and complex driving scenarios
  - Real-world data collection and training

Gaming & entertainment

- OpenAI Five for playing Dota 2
  - complex multi-agent environment
  - long-term planning (45+ minute games)
- DeepMind AlphaStar for playing StarCraft II
  - league-based training, population-based methods
  - partial observability challenges
  - human-level performance

Robotics

- Boston Dynamics
  - RL for dynamic locomotion
  - sim-to-real transfer
  - robust control policies
- Covariant - warehouse automation
  - RL for robotic picking and manipulation
  - real-world deployment in warehouses
  - continuous learning from experience

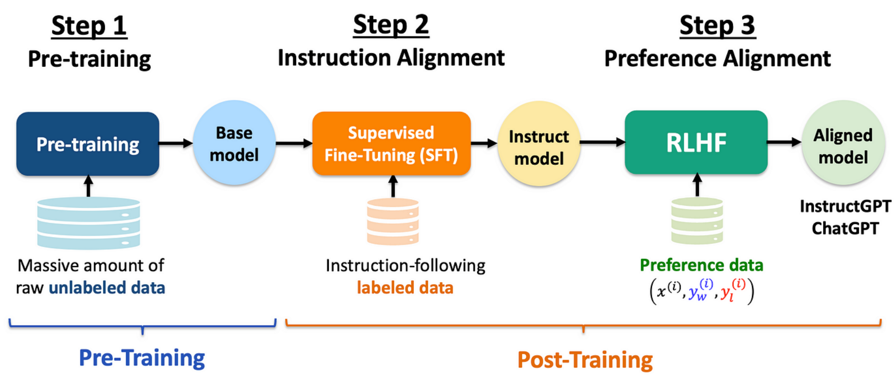
Finance & trading

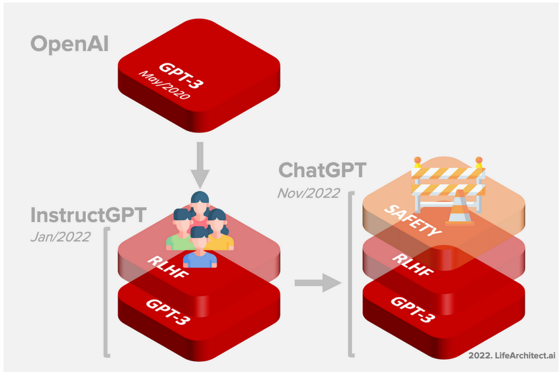
- JP Morgan Chase
  - algorithmic trading with RL
  - portfolio optimization
  - risk management
- Two Sigma, Renaissance Technologies
  - market making and execution
  - multi-agent trading environments

LLM & RL

RLHF - RL from human feedback

- ChatGPT, GPT-4 training pipelines
  - supervised fine-tuning - train on human demonstrations
  - reward model training - learn human preferences
- key components
  - reward model - predicts human preferences
  - KL penalty - prevents deviation from original model
  - constitutional AI - self-improvement through AI feedback





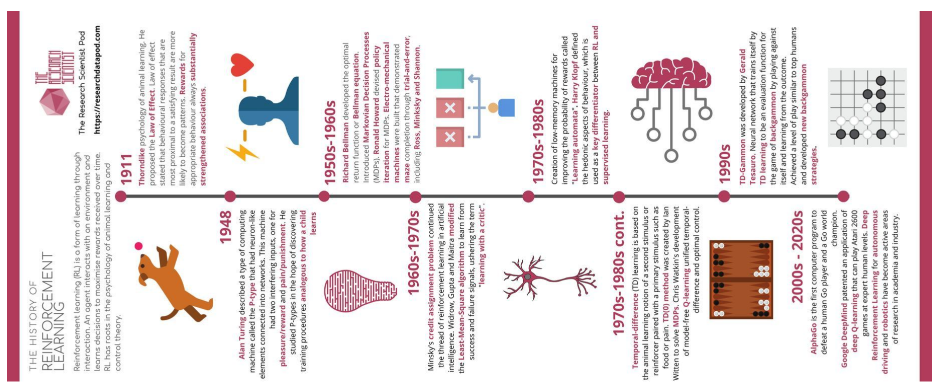
Applications in LLMs

- OpenAI - ChatGPT/GPT-4
  - RLHF for helpful, harmless, honest responses
  - massive scale PPO training
  - human preference learning
- Anthropic - Claude
  - constitutional AI methods
  - self-supervised preference learning
  - scalable oversight techniques

RL Evolution

Classical to modern RL

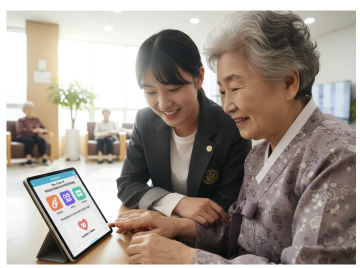
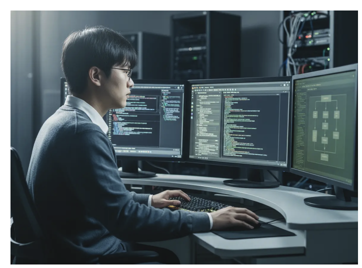
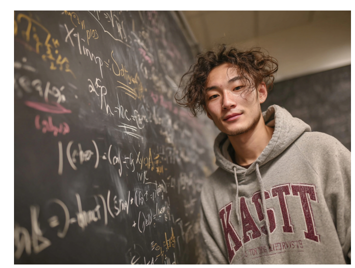
- key progressions
  - tabular $\rightarrow$ function approximation $\rightarrow$ DNN / model-free $\rightarrow$ model-based $\rightarrow$ hybrid
  - single agent $\rightarrow$ multi-agent $\rightarrow$ large-scale systems
- core principles *intact*
  - exploration vs exploitation trade-off / Bellman equations & Bellman optimality
  - policy improvement & evaluation / generalized policy iteration (GPI)
- modern additions
  - scale and compute power / human feedback integration
  - safety & robustness considerations / multi-modal and foundation models ($e.g.$, LLM)



Building Your Superpower

Students already building with AI - two paths, one future

- AI power user - domain experts using AI (majority)
  - 17-year-old high school student - passionate about helping elderly grandparents
    - built medication reminder app using Claude - no programming background
    - ". . . didn't learn to code. I learned to describe what grandparents need to AI."
  - college business student - interested in K-beauty industry
    - used Claude to analyze social media sentiment to predict K-beauty trend
    - "I understood Korean beauty culture. AI understood data patterns."
- AI expert - AI scientists & engineers & builders
  - computer science junior - 4 years studying math, algorithms, neural networks
    - fine-tuned open-source LLM for Korean medical terminology
    - collaborated with doctors (domain experts) on diagnostic tool

AI power user - domain expert AI-amplified

- who this is for?
  - you love literature, business, medicine, law, art, design, *etc.*
  - you find AI interesting as TOOL, not as end in itself
  - you get excited about domain problems, not algorithms
- what you'll do
  - deepen expertise in your chosen field (4+ years) - learn AI tools as power tools
  - use AI to amplify your domain work & compete on domain insight + AI leverage
- career examples
  - doctor using AI diagnostics, teacher using AI personalization
  - lawyer using AI research, artist using AI iteration
  - marketer using AI analytics, scientist using AI simulation

AI expert - researcher/scientist/engineer/developer

- who this is for?
  - you find algorithms, mathematics, systems *beautiful*, and read AI papers for fun
  - you want to work at AI labs being excited when new architectures are published
- what you'll do
  - deep study - mathematics, computer science, ML theory ($e.g.$, 4+ years)
  - understand neural networks, transformers to build and improve AI systems
  - *collaborate* w/ domain experts to apply your systems
- career examples
  - ML Engineer working for (tech) companies
  - AI Researcher in academia or industry labs
  - research scientist, robotics engineer, computer vision specialist

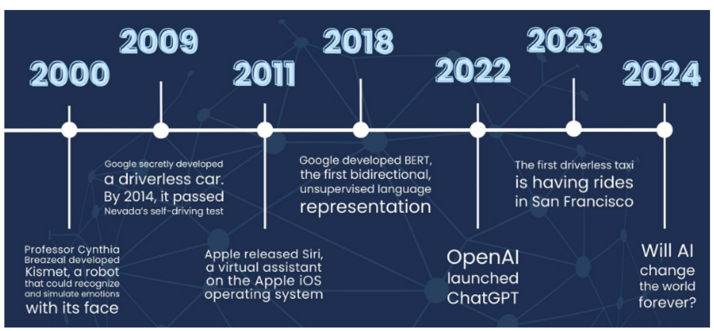


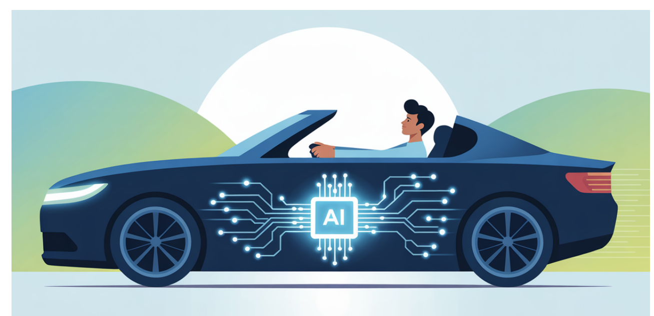

BIG shifts in AI landscape

- old assumption (2015 – 2020)
  - AI will be built by tech companies, everyone else will be disrupted
  - lots of tech companies will compete for best models/products
- new reality (2024 –)
  - AI is commoditized tool, domain expertise is where value accrues
  - only handful of companies can develop cutting-edge foundation models

Domain expert revolution

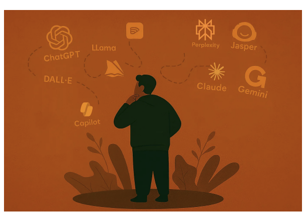
- why domain experts are winning
  - problem identification requires domain knowledge
  - evaluation requires domain judgment
  - trust requires human domain expertise
  - context requires cultural/domain fluency
- pattern across industries
  - best medical AI applications → built by doctors collaborating with AI engineers
  - best legal AI tools → built by lawyers collaborating with AI engineers
  - best educational AI → built by teachers collaborating with AI engineers
  - domain expert leads & AI engineer supports (not the reverse)

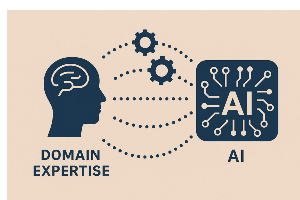




AI power user - what you actually need to learn

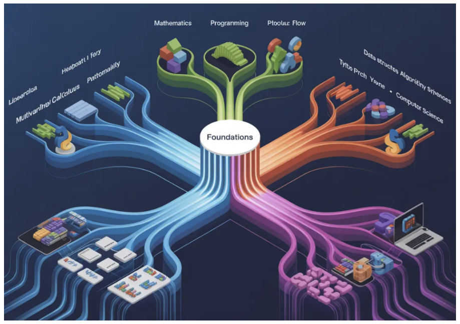
- tool awareness
  - what - know which AI tools exist and their capabilities
  - examples - ChatGPT for brainstorming, Claude for research, NotebookLM for synthesis
  - how - YouTube tutorials, free trials, experimentation
- right questions to throw - interactive way
  - what - communicate effectively with AI using domain knowledge
  - why it works - your domain knowledge makes prompts effective
  - how - practice + domain expertise
- tool integration
  - what - connect AI tools to your workflow, build custom GPTs, use APIs (no coding)
  - how - no-code tools + tutorials

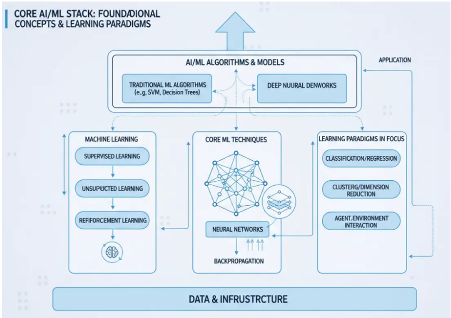




AI expert path - how to become AI scientist/researcher/engineer/practitioner

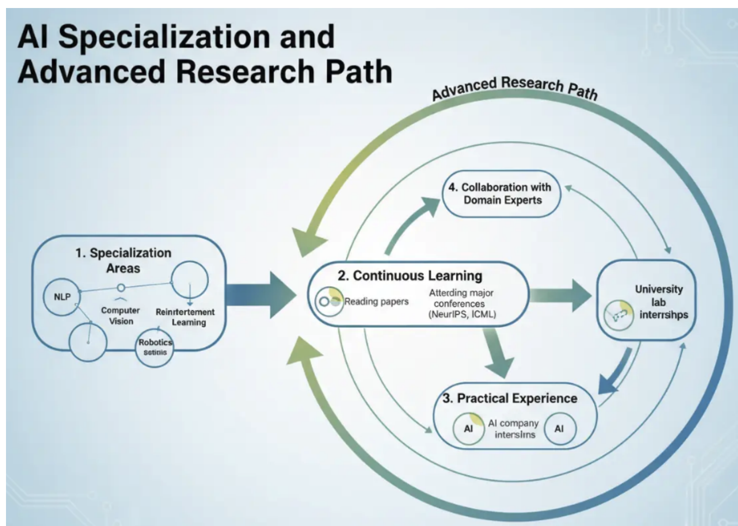
- foundation
  - mathematics - linear algebra, multivariable calculus, probability theory, optimization
  - programming - python w/ PyTorch, TensorFlow, . . .
  - computer science - data structures, algorithms, systems
- core AI/ML
  - machine learning (ML) - supervised/unsupervised learning, reinforcement learning, neural networks, backpropagation
  - deep learning (DL) - CNNs, RNNs, Transformers, attention mechanisms
  - practical projects - competitions, replicate papers, contribute to open source





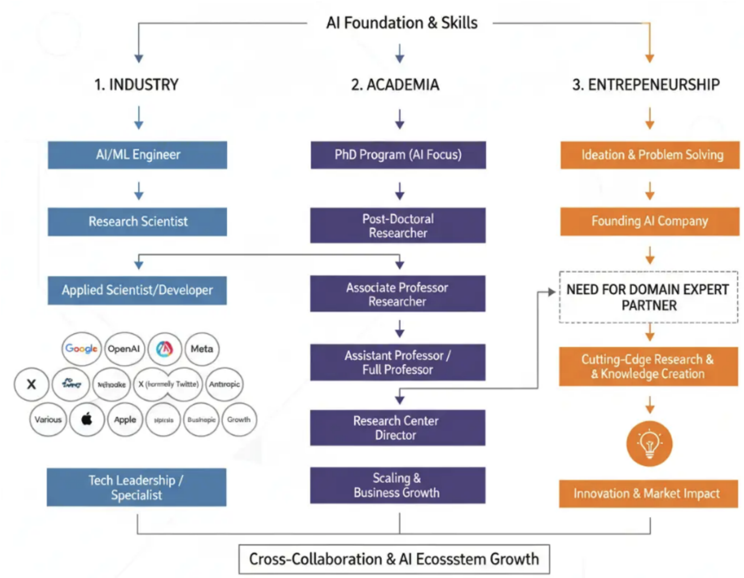
AI expert path - how to become AI scientist/researcher/engineer/practitioner

- specialization
  - choose area (or not!) - NLP, computer vision (CV), robotics, *etc.*
  - read papers - arxiv, attend conferences (*e.g.*, NeurIPS, ICML, ICLR, CVPR, *etc.*)
  - research experience - lab work at university, internships at AI companies
  - collaboration - work with domain experts on real problems



AI expert path - how to become AI scientist/researcher/engineer/practitioner

- career paths
  - industry - Google, OpenAI, Meta, Apple, X, Anthropic, and numerous startups
  - academia - PhD $\rightarrow$ professor / research center
  - entrepreneurship - found AI company (but need domain expert partner!)



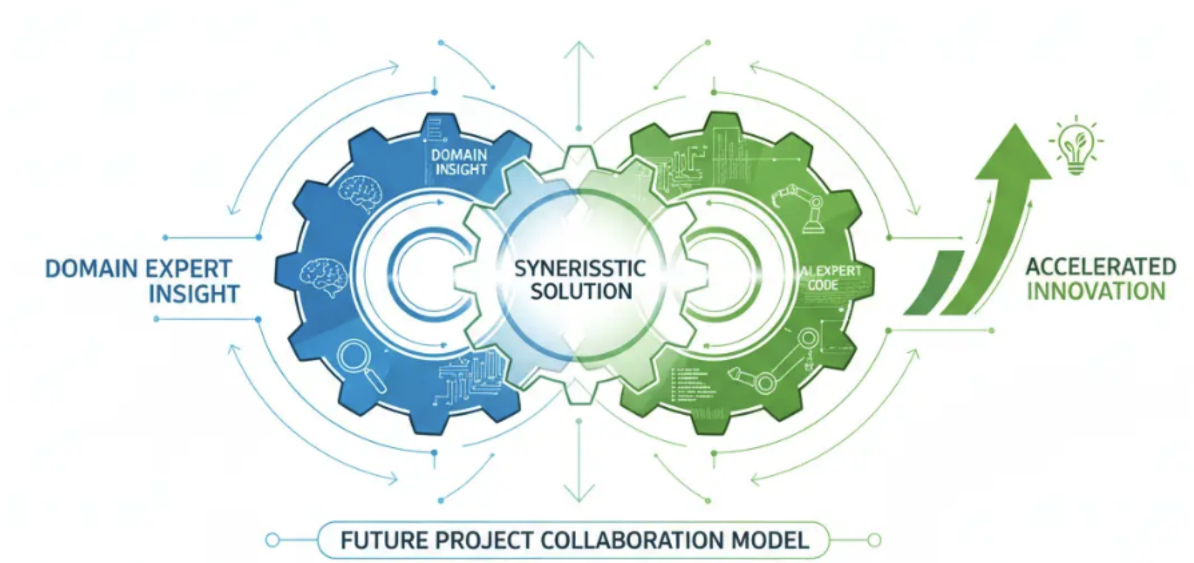
Powerful combination - domain expert + AI expert collaboration

The magic happens at the intersection!

- case study 1 - AlphaFold (Protein Folding)
  - AI engineers - build transformer-based neural network, optimized training
  - structural biologists - identify protein folding as THE problem, validated outputs
  - result - 50-year problem solved - neither could do it alone
  - both Demis Hassabis (CEO @ Google DeepMind) & John Jumper (biochemistry background) got Nobel Prizes in chemistry in 2024!
- case study 2 - cancer diagnostic AI
  - AI engineer role - build model, handle large medical imaging datasets
  - oncologist role - label training data correctly, evaluate clinical relevance
  - result - 30% improvement in early detection
  - who makes final diagnosis - always the doctor!

Powerful combination - domain expert + AI expert collaboration

- case study 3 - your future project
  - domain expert & AI super user- identify problem in your field you understand deeply
  - AI expert - build custom solution beyond available tools
  - result - breakthrough that neither could achieve alone
  - who drives vision - TOGETHER!



Things AI can't do for you - choose who you'll become

- your technical superpower (Domain $\times$ AI) is only as good as *your moral compass*
- why this matters more than ever
  - AI amplifies whatever you choose to do - good intentions OR bad ones
  - history shows us - brilliant minds + no moral foundation = catastrophe
  - your skills will make you powerful; your values determine what you do with that power
- virtues that actually matter in the long run
  - *integrity* - doing the right thing (even) when no one's watching
  - *empathy* - genuinely caring about people; your technology will affect them
  - *service* - building things that help others, not just things that benefit you
  - *morality* - having inner compass that guides you beyond what's legal or profitable

Selected References & Sources

Selected references & sources

- Robert H. Kane "Quest for Meaning: Values, Ethics, and the Modern Experience" 2013
- Michael J. Sandel "Justice: What's the Right Thing to Do?" 2009
- Daniel Kahneman "Thinking, Fast and Slow" 2011
- Yuval Noah Harari "Sapiens: A Brief History of Humankind" 2014
- M. Shanahan "Talking About Large Language Models" 2022
- A.Y. Halevry, P. Norvig, and F. Pereira "Unreasonable Effectiveness of Data" 2009
- A. Vaswani, et al. "Attention is all you need" @ NeurIPS 2017
- S. Yin, et. al. "A Survey on Multimodal LLMs" 2023
- Chris Miller "Chip War: The Fight for the World's Most Critical Technology" 2022

- CEOs, CTOs, CFOs, COOs, CMOs & CCOs @ startup companies in Silicon Valley
- VCs on Sand Hill Road - Palo Alto, Menlo Park, Woodside in California, USA

References

References

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Bradford Books, 2nd edition, 2018.

Thank You