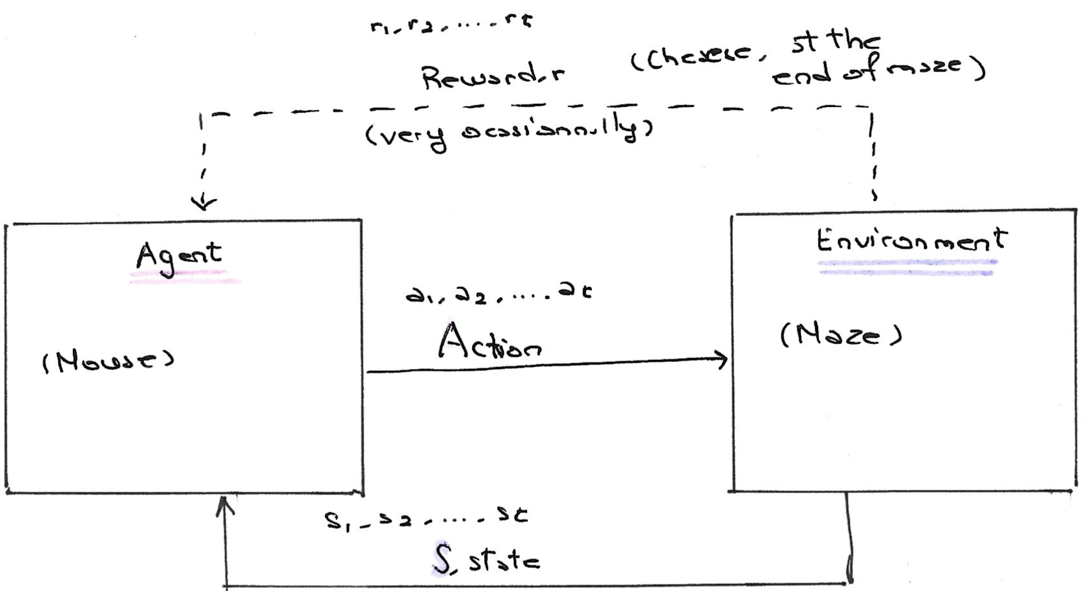
INTRODUCTION TO

Reinforcement Learning.

- A framework for Learning how to Interact with the Environment from Experience.
- Biologically - Inspired Idea (Trial/Error/Experiences)

Reinforcement Learning Overview

→ NULL most of the time



Note: Ø Bird-Eye view of the Maze (only present and past: where it is/was)

- If every correct turn, it got a piece of cheese (reward), this is supervised learning. Rewards = LABELS.
- Because the reward is time-delayed (At the end of maze), not depending on each move of the mouse, Reward: TIME-DELAYED LABEL
  ↳ Semi-supervised Learning.
- Since we have less labels, requires much more data to train these agents and more trial/error.
  ↳ Harder optimization problem.

⇒ Another example:

Agent: Chess
Environment: Rules of the game

Mathematics of RL.

• Example: Chess:

→ Challenge: design a policy of what actions to take given a state s to max. my chance of getting a future reward.

↳ $\boxed{\text{Policy } \pi(s,a)}$: $\mathbb{P}\{A=a \mid S=s\}$, ∅ deterministic.

→ Only reward once you beat chess. If I lose, do I remove the entire sequence of actions? How do you figure which actions are good/bad?

$\boxed{\text{VALUE}}$: $V_\pi(s) = \mathbb{E}\left\{ \sum_t \boxed{\gamma}^t r_t \mid s_0 = s \right\}$

↳ Discount Rate

↳ Expectation of how much reward I'll get in the future if I start in that state s and I enact that policy

• Discount Rate $0 \le \gamma \le 1$

→ $\boxed{\text{Goal}}$: Optimize $\pi(s,a)$, the policy, to maximize future rewards.

Markov Decision Process.

→ Environment contains a stochastic component

→ $\mathbb{P}$ of going from current state/action to the next.

Credit Assignment Problem.

→ Since the rewards are sparsed ⇒ very difficult to find actions that are good/bad

→ 1960s Minsky

→ Dense vs. Sparse Rewards

→ Sample Efficiency — Many trials/examples/games.

→ Reward Shaping.

Optimization Techniques for RL.

- → Differential Programming.
- → Monte-Carlo
- → Temporal Difference (Model free)
- → Bellman 1957
- → Exploration vs. Exploitation.
- → Policy Iteration.
- → Gradient Descent, Evolutionary Opt, Simulated Annealing.

$V(S_t) = \sum r^t \Leftrightarrow$ future rewards

$Q(a_t, s_t) = r_{t+1} + \sum r^t$

$= r_{t+1} + V(s_t)$

Q-Learning.

- $Q(s,a)$: quality of state / Action Pair

$$Q^{update}(s_t, a_t) = Q^{old}(s_t, a_t) + \alpha \left\{ r_t + \gamma \max_a Q(s_{t+1}, a) - Q^{old}(s_t, a_t) \right\}$$

→ Given a state, we can choose the action that has the highest $\boxed{\text{Quality}}$.

$\pi_\theta(a|s) \Rightarrow Q_\phi(\pi(s))$

Hindsight Replay.

- Data efficient
- Save «mistakes» for the future, ~~reward~~ sample efficient.

Overview of Methods

2

Model-based RL → observed on history, only based on current/future states.

Markov Decision Process $P(s', s, a)$
- → Policy Iteration $\pi_\theta(s,a)$
- → Value Iteration $V(s)$

Satisfies...

Dynamic Programming & Bellman Optimality.

Non Linear Dynamics

$\frac{d}{dt} x = f(x(t), u(t), t) \, dt$

→ Optimal Control & HJB
⇓
Hamilton/Jacobi/Bellman

⇓

Brute force Search

Curse of Dimensionality.

Actor Critic

Deep MPC

DEEP L.

Model-free RL

Uses Quality function
⇓
DQN

$\boxed{\text{Off Policy}}$ (*) vs $\boxed{\text{On Policy}}$

Gradient-Free

Off Policy:
- Random move occasionally ⇒ helpful for learning the environment.

$Q(s,a)$

→ Q-Learning. (faster)

On Policy:
- Always use the best policy every game

TD (Temporal Δ)

TD(0) ... TD(∞) ≡ MC

→ SARSA (conservative)

Gradient-based

Deep Policy Network

$\theta^{new} = \theta^{old} + \alpha \nabla_\theta R_{\Sigma,\theta}$

→ Policy Gradient Optimization

MODEL-BASED RL : MARKOV DECISION PROCESS

- We assume that we have a model for how the env. works / how the system works. follows the MDP. ⇒ for $P(s', s, a)$. We also assume a model for the reward $(R(s', s, a))$.

- $R(s', s, a) = \mathbb{P}\{r_{k+1} \mid s_{k+1} = s', s_k = s, a_k = a\}$
- $P(s', s, a) = \mathbb{P}\{s_{k+1} = s' \mid s_k = s, a_k = a\}$

⇒ Assumption: knows the rule of the game (Ex: knows how to checkmate someone in a chess game).

⇒ Dynamic Programming instead of Brute Force Search.

Value Function: $V_\pi(s) = \mathbb{E}\left\{\sum_k \gamma^k r_k \mid s_0 = s\right\}$ → Expected value of future rewards given an initial state $s_0 = s$.

Optimized

$$\rightarrow V(s) = \max_\pi \mathbb{E}\left\{\sum_{k=0}^{\infty} \gamma^k r_k \mid s_0 = s\right\}$$

$$= \max_\pi \mathbb{E}\left\{\underbrace{r_0}_{\text{current}} + \underbrace{\sum_{k=1}^{\infty} \gamma^k r_k \mid s_1 = s'}_{\text{future}}\right\} \iff \max_\pi \mathbb{E}\left[r_0 + \gamma V(s')\right]$$

$$\boxed{V(s) = \max_\pi \mathbb{E}\left[r_0 + \gamma V(s')\right]}$$ ⇒ Bellman's Equation. (On Policy)

$$\Updownarrow$$

$$\boxed{\pi = \arg\max_\pi \mathbb{E}\left[r_0 + \gamma V(s')\right]}$$

→ Bellman: how to solve multi-step optimization problems by breaking into smaller recursive sub-problems. TD: List out all sub-problems, until you solve the whole thing (Table Approach)

→ Bottom-Up vs Top-Down Approach. BU: Always start with smaller sub-problems (Reward guaranteed → 1 step / 2 steps ... Ahead)

Value Iteration ⇒ Dynamic Programming.

Discount Rate.

$$V(s) = \max_a \left\{\sum_{s'} P(s' \mid s, a)\left[R(s', s, a) + \gamma V(s')\right]\right\}$$ → To get this optimal future reward

$\max_a$ ⇓ Find the best Action

$R(s', s, a)$ ⇓ My current reward

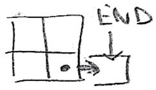
$V(s')$ ⇓ Next step given that Action a.

Assumes that $P(s' \mid s, a)$, $R(s', s, a)$ and $V(s')$ are given.

3248 AA

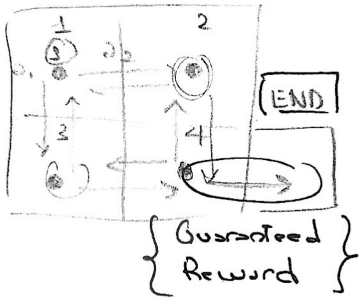
TD3 ⇒ continuous actions
DQN ⇒ discrete actions

- Create a Table of all possible states. (Initialize 0).
- Randomly take a state s.
  - ↳ Maximize value function given an optimized Action.
- Take a new state s and repeat the same process.



↳ Iterate until the value function converges.

$$\pi(s,a) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s,a)\left[R(s',s,a) + \gamma V(s')\right]$$



Policy Iteration

→ directly onto the rules of the game (not to maximize the reward based on what actions to take)

- $V_\pi(s) = \mathbb{E}\left[R(s', s, \boxed{\pi(s)}) + \gamma V_\pi(s')\right] = \sum_{s'} P(s'|s,\pi(s)) \cdot \left[R(s',s,\pi(s)) + \gamma V_\pi(s')\right]$
- $\pi(s) = \underset{a}{\operatorname{argmax}}\ \mathbb{E}\left[R(s',s,\pi(s)) + \gamma V_\pi(s')\right]$

⇒ Typically converges in fewer iterations.

Quality Function.

$$Q(s,a) = \mathbb{E}\left[R(s',s,a) + \gamma V(s')\right] = \sum_{s'} P(s'|s,a) \cdot \left[R(s',s,a) + \gamma V(s')\right]$$

↳ we don't need a new model for the future states s' ⇒ implicit in the Q function
⇒ becomes Model-free.

Therefore:

$$\boxed{V(s) = \max_a Q(s,a)}$$

⇒ Expected value of future rewards OR the maximized quality function given an action a and a state s.

$$\boxed{\pi(s,a) = \underset{a}{\operatorname{argmax}}\ Q(s,a)}$$

↳ Policy optimized: the action a that maximizes our future rewards (the quality function).

4 Q - LEARNING (GRADIENT - FREE)

· $Q(s,a)$: quality of state/action pair

⇒ $V(s)$: value of being in a current state s ⇒ assuming you take the best Action.

⇒ $Q(s,a)$: for any action a I may take (more information than the value function)

$$Q(s,a) = \mathbb{E}\left[\underbrace{R(s',s,a)}_{\text{current reward (from current state s to s')}} + \gamma \underbrace{V(s')}_{\text{future rewards}}\right] = \sum_{s'} P(s'|s,a)\left[R(s',s,a) + \gamma V(s')\right]$$

$$\Updownarrow$$

$$\left\{ \begin{aligned} V(s) &= \max_a Q(s,a) \\ \pi(s,a) &= \underset{a}{\operatorname{argmax}}\, Q(s,a) \end{aligned} \right\}$$

⇒ Generally, we don't know the models for $P(s'|s,a)$ and $R(s',s,a)$.
⇒ Stuck in Trial and Error Learning.

Monte-Carlo Learning (Model-Free)

n — # steps. (Episodes) ⇒ for one state.

· $R_\Sigma = \sum_{k=1}^{n} \gamma^k r_k$ TOTAL REWARD OVER EPISODE

→ { Estimated Reward (V^{old})
Actual Reward (R_Σ)

$$\Rightarrow V^{new}(s_k) = V^{old}(s_k) + \frac{1}{n}\{R_\Sigma - V^{old}(s_k)\} \quad \forall k \in \{1, \dots, n\}$$

$$\Rightarrow Q^{new}(s_k, a_k) = Q^{old}(s_k, a_k) + \frac{1}{n}\{R_\Sigma - Q^{old}(s_k, a_k)\}$$

⇒ Not recommended, a lot of episodes required.
However, no biases.

↳ The right action could've happened at any point in the past leading up to the reward now.

↳ TD: there's an optimal time (past) that is well correlated to the current reward.

Temporal Difference Learning : TD(0) weighting only a few past (1 here ...)

- $V(s_k) = \mathbb{E}[r_k + \gamma V(s_{k+1})]$ ⇒ Bellman's Optimality.

$$\Rightarrow V^{new}(s_k) = V^{old}(s_k) + \alpha \{ \overbrace{\underbrace{r_k + \gamma V^{old}(s_{k+1})}_{\text{TD Target, Estimate } R_\Sigma \text{, One } \Delta t \text{ delay.}} - V^{old}(s_k)}^{\text{TD Error}} \}$$

↳ $$R_\Sigma^{(n)} = r_k + \gamma r_{k+1} + \gamma^2 r_{k+2} + \dots + \gamma^n r_{k+n} + \gamma^{n+1} V(s_{k+n+1})$$

$$= \sum_{j=0}^{n} \gamma^j r_{k+j} + \gamma^{n+1} V(s_{k+n+1}) \quad \Rightarrow \text{for TD(N)}$$

If $N \to \infty$: TD(∞) = Monte-Carlo

$\boxed{\text{TD-}\lambda}$

where $0 \le \lambda \le 1$

$$R_\Sigma^\lambda = (1-\lambda) \sum_{n=0}^{\infty} \lambda^{n-1} R_\Sigma^{(n)}$$

$$\Rightarrow V^{new}(s_k) = V^{old}(s_k) + \alpha \{ R_\Sigma^\lambda - V^{old}(s_k) \}$$

Q-Learning.

- Temporal Difference on Quality functions... ↳ NOT on the policy functions...

$$Q^{new}(s_k, a_k) = Q^{old}(s_k, a_k) + \alpha \{ \overbrace{\underbrace{r_k + \gamma \max_a Q(s_{k+1}, a)}_{\text{TD Estimate } R_\Sigma \text{ (Target)}} - Q^{old}(s_k, a_k)}^{\text{TD Error}} \}$$

⇒ off-Policy TD(0) learning of the quality function Q.

⇒ You can still learn even if you take sub-optimal actions a.

SARSA : State - Action - Reward - State - Action

$$Q^{new}(s_k, a_k) = Q^{old}(s_k, a_k) + \alpha \{ \underbrace{r_k + \gamma Q^{old}(s_{k+1}, a_{k+1})}_{\text{Need to } \Uparrow} - Q^{old}(s_k, a_k) \}$$

we want to converge

$\boxed{\text{ON-POLICY}}$

⇒ On-Policy TD(0) learning on the $Q(s_k, a_k)$.

- Q-Learning : $\max_a Q(s_{k+1}, a)$ ⇒ Off-Policy
  - ① Better / Faster Learning; Higher Variance
  - ② Can Learn from Imitation and Exp. Replay.

⇒ Epsilon-Greedy: $\varepsilon \uparrow$ → cools down to 0

ε: % of exploration (random actions) into off policy strategies.

- SARSA : $Q^{old}(s_{k+1}, a_{k+1})$ ⇒ On-Policy.
  - ① Often Safer
  - ② Better total Reward while Learning.

⇒ Q-Learning and SARSA: Approximate Dynamic Programming.

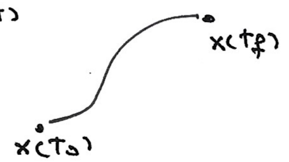
↳ Q-Learning for Deep RL.

5 OPTIMAL NON-LINEAR CONTROL

$\frac{d}{dt} x = f(x(t), u(t))\, dt$, where : x: info that charactirizes the state of my system

(Ex: pendulum ⇒ position of the cart, ↳ velocity, etc.)

- f: dynamics of that system
- Goal: design a control function $u(t)$ to follow a state $x(t)$ to minimize the cost J.



$$\Rightarrow J(x(t), u(t), t_0, t_f) = Q(x(t_f), t_f) + \int_{t_0}^{t_f} \mathcal{L}(x(\tau), u(\tau))\, d\tau$$

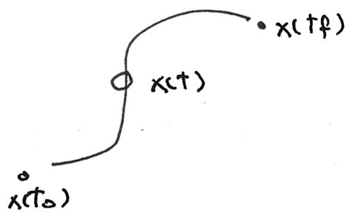
↳ Ex: $x(\tau)^2 + u(\tau)^2$ for example.

$$\Rightarrow V(x(t_0), t_0, t_f) = \min_{u(t)} J(x(t), u(t), t_0, t_f)$$

$$\boxed{-\frac{\partial V}{\partial t} = \min_{u(t)} \left\{ \left(\frac{\partial V}{\partial x}\right)^T f(x(t), u(t)) + \mathcal{L}(x(t), u(t)) \right\}}$$

HAMILTON JACOBIAN BELLMAN (HJB) EQUATION.

$$V(x(t_0), t_0, t_f) = \underbrace{V(x(t_0), t_0, t) + V(x(t), t, t_f)}_{\text{Bellman's Optimality.}}$$



Deriving HJB Equation

$$\frac{d}{dt} V(x(t), t, t_f) = \frac{\partial V}{\partial t} + \left[\frac{\partial V}{\partial x}\right]^T \frac{dx}{dt}$$

$$= \min_{u(t)} \frac{d}{dt} \left\{ \int_0^{t_f} \mathcal{L}(x(\tau), u(\tau))\, d\tau + Q(x(t_f), t_f) \right\}$$

$$= \min_{u(t)} \left\{ \underbrace{\frac{d}{dt} \int_0^{\tau_f} \mathcal{L}(x(\tau), u(\tau))\, d\tau}_{-\mathcal{L}(x(t), u(t))} \right\}$$

$$\Rightarrow -\frac{\partial V}{\partial t} = \min_{u(t)} \left\{ \left(\frac{\partial V}{\partial x}\right)^T f(x,u) + \mathcal{L}(x,u) \right\}$$

Discrete - Time HJB

$$x_{k+1} = F(x_k, u_k)$$

$$J(x_0, \{u_k\}_{k=0,n}^{n}) = \sum_{k=0}^{n} \mathcal{L}(u_k, x_k) + Q(x_n, t_n)$$

$$V(x_0, n) = \min_{\{u_k\}_{k=0}^{n}} J(x_0, \{u_k\}_{k=0,n}^{n})$$

$$V(x_0, n) = V(x_0, k) + V(x_k, n-k) \quad \forall k \in (0, n) \quad \Rightarrow \text{Optimality Condition}$$

$$\Rightarrow V(x_k, n) = \min_{u_k} \left\{ \mathcal{L}(x_k, u_k) + V(x_{k+1}, n-1) \right\}, \text{ where } x_{k+1} = F(x_k, u_k)$$

current cost

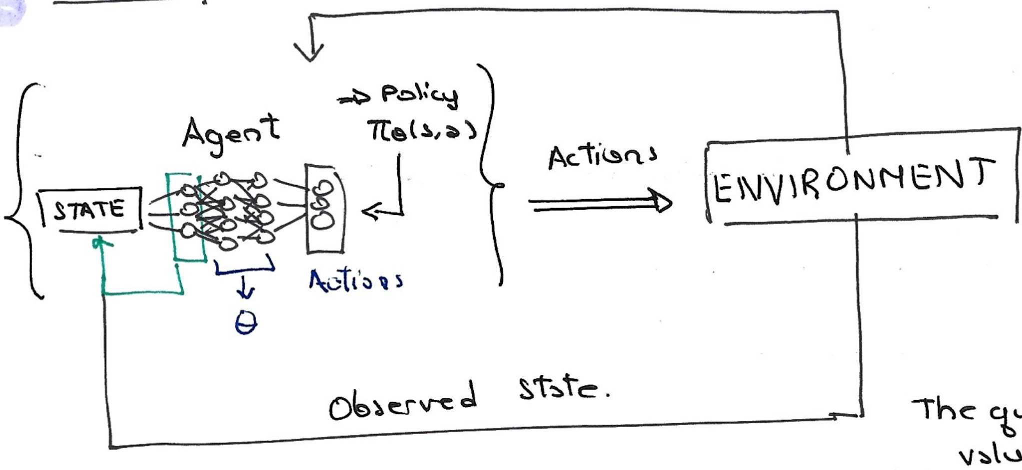
future steps (its value function)

optimized.

Therefore...

$$\begin{cases} V(x) = \min_u \left(\mathcal{L}(x,u) + V(F(x,u)) \right) \\ \pi(x) = \arg\min_u \left(\mathcal{L}(x,u) + V(F(x,u)) \right) \end{cases}$$

6 DEEP RL (IMPORTANT!!!)



Value ⇒ V function

$$\mathbb{E}\left[\sum_t \gamma^t r_t \mid s_0 = s\right]$$

Q-Learning:

⇒ Q: function that combines value and policy.

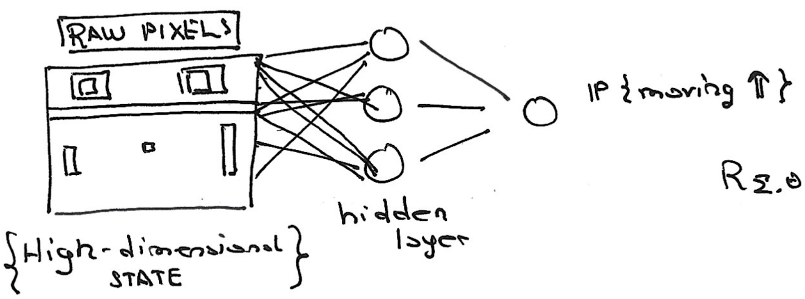
The quality function tells how valuable your actions are while strengthning the policy.

• However, the inconvenients related to RL are still influenced to Q-Learning.

7 DEEP RL METHODS (IMPORTANT!!!)

1. Deep Policy Network

⇒ $\pi_\theta(s,a)$: parametrized as a neural network (weights + bias)



{through back propagation}

Policy $\pi_\theta(s,a)$

$$\Rightarrow \theta^{new} = \theta^{old} + \alpha \nabla_\theta R_{\Sigma,\theta}$$

$R_{\Sigma,\theta}$: cumulated expected reward (we can use a value network for example)...

Policy Gradient Opt: (more general than NN)

$$R_{\Sigma,\theta} = \boxed{\sum_{s \in S}} \mu_\theta(s) \boxed{\sum_{a \in A}} \pi_\theta(s,a) Q(s,a)$$

⇒ quality

↳ probability

Randomized

⇓ Expected prob. of having s

{Assymptotic distribution}

↳ Randomized

⇒ Consistent with $R_{\Sigma,\theta}$

$$\nabla_\theta R_{\Sigma,\theta} = \sum_{s \in S} \mu_\theta(s) \sum_{a \in A} Q(s,a) \nabla_\theta \pi_\theta(s,a)$$

⇓ Move towards MAX Rewards!

↳ why not differentiate this variable?

Assumed stationary in an ergodic environment

$$\nabla_\theta R_{\Sigma,\theta} = \sum_{s\in S} \mu_\theta(s) \sum_{a\in A} \pi_\theta(s,a)\, Q(s,a) \cdot \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)}$$

$$= \sum_{s\in S} \mu_\theta(s) \sum_{a\in A} \pi_\theta(s,a)\, Q(s,a)\, \nabla_\theta \log\{\pi_\theta(s,a)\}$$

$$= \mathbb{E}\left[Q(s,a) \underbrace{\nabla_\theta \log\{\pi_\theta(s,a)\}}_{} \right]$$

↳ back propagation...

$$\theta^{new} = \theta^{old} + \alpha \nabla_\theta R_{\Sigma,\theta}$$

$$\left\{ \begin{array}{l} \frac{(x^2)'}{x^2} = \frac{2x}{x^2} = \frac{2}{x} \\ \log\{x^2\} = \frac{1}{x^2} \cdot (x^2)' \end{array} \right.$$

Example with x^2

---

2. Deep Q-Learning (DQN)

- Off-Policy TD(.)

$$Q^{new}(s_k, a_k) = Q^{old}(s_k, a_k) + \alpha\{r_k + \gamma \max_a Q(s_{k+1}, a) - Q^{old}(s_k, a_k)\}$$

$$\Rightarrow Q(s,a) \approx Q(s,a,\theta)$$ ⇒ parameterize Q Function with NN.

⇓

↓ dimension (Extract low dimensional features)

- NN Cost Function: $\mathcal{L} = \mathbb{E}\left[\{r_k + \gamma \max_a Q(s_{k+1}, a_{k+1}, \theta) - Q(s_k, a_k, \theta)\}^2 \right]$

minimize temporal diff-error by optimizing θ ~~a~~ in NN.

3. Advantage Network (DDQN: Deep-Dueling)

- $Q(s,a,\theta) = V(s,\theta_1) + A(s,a,\theta_2)$

↓ Value Network. (Explanation from state)

↳ Advantage Network (how effective the action is in that state).

- Splits into 2 networks

4. Actor-Critic Network

- ACTOR: Policy based → $\pi(s,a) = \pi(s,a,\theta)$
- CRITIC: VALue based → $V(s_k) = \mathbb{E}[r_k + \gamma V(s_{k+1})]$

$$\underbrace{\theta_{k+1} = \theta_k + \alpha}_{\text{Policy}} \underbrace{\{r_k + \gamma V(s_{k+1}) - V(s_k)\}}_{\text{Value (Error)}}$$

⇒ Best of Both worlds

⇓

Using TD signals from critic to update policy parameters

5. Advantage Actor-Critic Network

- ACTOR: Deep Policy Network $\rightarrow$ $\pi(s,a) \approx \pi(s,a,\theta)$
- CRITIC: Deep Dueling Q Network $\rightarrow$ $Q(s_k, a_k, \theta_2)$

$$\theta_{k+1} = \underbrace{\theta_k + \alpha \nabla_\theta \{ (\log \pi(s_k, a_k, \theta))}_{\text{Policy}} \cdot \underbrace{Q(s_k, a_k, \theta_2)}_{\text{DDQN: quality.}} \} \Rightarrow \text{UPDATE.}$$

DEEP RL IN FLUID DYNAMICS

- Biological RL: the rewards are internal (release dopamine)
- Usually, rewards are in the environment.