1 Fondamenti del Reinforcement Learning

1.1 Il Processo Decisionale di Markov (MDP)

Il Reinforcement Learning (RL) è un'area del machine learning che si occupa di come un agente debba compiere azioni in un ambiente al fine di massimizzare una nozione di ricompensa cumulativa. Il framework matematico che formalizza questo problema di apprendimento sequenziale è il **Processo Decisionale di Markov (Markov Decision Process, MDP)**, come definito nel testo di riferimento di Sutton e Barto [1]. Un MDP è una tupla $(\mathcal{S}, \mathcal{A}, p, R, \gamma)$, dove:

- $\mathcal{S}$ è l'insieme di tutti i possibili stati dell'ambiente.
- $\mathcal{A}$ è l'insieme di tutte le azioni che l'agente può compiere.
- $p(s', r|s, a) = \Pr(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$ è la funzione di dinamica dell'ambiente. Essa definisce la probabilità di transire allo stato s' e ricevere la ricompensa r, dato che l'agente si trova nello stato s e compie l'azione a. Questa funzione soddisfa la **proprietà di Markov**.
- R è l'insieme delle possibili ricompense.
- $\gamma \in [0, 1)$ è il fattore di sconto, che determina l'importanza delle ricompense future rispetto a quelle immediate.

La **proprietà di Markov** è il fondamento di questo framework e afferma che "il futuro è indipendente dal passato, dato il presente". In termini formali, la probabilità dello stato e della ricompensa successivi dipende *esclusivamente* dallo stato e dall'azione correnti, S_t e A_t, e non dall'intera sequenza di stati, azioni e ricompense che li hanno preceduti.

- **Esempio Markoviano**: In una partita a scacchi, la configurazione attuale della scacchiera è uno stato che soddisfa la proprietà di Markov. Le mosse future ottimali dipendono solo da dove si trovano i pezzi ora, non da come sono arrivati in quella posizione.
- **Esempio non-Markoviano**: Un agente che naviga con un sensore di posizione rumoroso. Per stimare la sua vera posizione attuale (lo stato), l'agente potrebbe aver bisogno dell'intera cronologia delle sue osservazioni e azioni, non solo dell'ultima lettura. In questo caso, lo stato osservato non è sufficiente e il processo non è Markoviano.

L'interazione si svolge in una sequenza di istanti di tempo discreti, $t = 0, 1, 2, \ldots$. Ad ogni istante t, l'agente osserva lo stato S_t e seleziona un'azione A_t. L'ambiente risponde con una ricompensa R_{t+1} e un nuovo stato S_{t+1}.

1.2 Obiettivi, Ricompense e Ritorni

L'obiettivo dell'agente è massimizzare il **ritorno** (G_t), definito come la somma scontata delle ricompense future. Questa somma infinita è garantita convergere a un valore finito a condizione che il fattore di sconto γ sia strettamente minore di 1 e che le ricompense siano

limitate (o almeno abbiano un valore atteso finito). Formalmente, il ritorno all'istante t è definito come:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \qquad \text{assumendo } \mathbb{E}[|R_t|] < \infty \text{ e } \gamma < 1 \text{ affinchè la serie converga.}$$

Una **politica** $\pi(a|s)$ è una distribuzione di probabilità sulle azioni dato uno stato. L'obiettivo ultimo del RL è trovare una politica ottimale π_* che massimizzi il ritorno atteso da ogni stato.

1.3 Value Functions e le Equazioni di Bellman

Per valutare l'efficacia di una politica, quasi tutti gli algoritmi di RL stimano delle **Value Functions** [1], che esprimono il valore a lungo termine di trovarsi in uno stato o di compiere un'azione in uno stato.

- La **state-value function** $v_\pi(s)$ è il ritorno atteso partendo dallo stato s e seguendo la politica π: $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$.
- La **action-value function** $q_\pi(s, a)$ è il ritorno atteso partendo da s, compiendo l'azione a e poi seguendo π: $q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$.

Queste funzioni soddisfano una fondamentale relazione ricorsiva nota come **Equazione di Bellman**. Per v_π, essa decompone il valore di uno stato nella ricompensa immediata attesa e nel valore scontato degli stati successori:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')].$$

Le funzioni valore ottimali, v_* e q_*, soddisfano l'**Equazione di Bellman di Ottimalità**, che per q_* afferma che il valore dell'azione ottimale deve essere uguale al ritorno atteso che si ottiene compiendo quell'azione e proseguendo poi in modo ottimale:

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')].$$

La **Advantage function** $A_\pi(s, a) = q_\pi(s, a) - v_\pi(s)$ quantifica il vantaggio relativo di un'azione a rispetto al comportamento medio della politica π nello stato s.

2 Il Teorema del Policy Gradient: Derivazione Dettagliata

I metodi Policy Gradient ottimizzano direttamente una politica parametrizzata $\pi_\theta(a|s)$. Il loro fondamento è il **Policy Gradient Theorem** [2], che fornisce un'espressione per il gradiente della funzione obiettivo $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[G_0]$ rispetto ai parametri θ.
La derivazione è un passaggio chiave che merita di essere esposto in dettaglio.

1. **Esplicitare l'aspettativa**: L'obiettivo è il valore atteso del ritorno su tutte le possibili traiettorie τ, pesato dalla loro probabilità sotto la politica π_θ.

$$\nabla_\theta J(\theta) = \nabla_\theta \int_\tau P(\tau|\theta) G_0(\tau) d\tau.$$

2. **Applicare il *log-derivative trick***: Per portare l'operatore di gradiente all'interno dell'integrale, usiamo l'identità $\nabla_x f(x) = f(x)\nabla_x \log f(x)$. Questo ci permette di esprimere il gradiente della probabilità $P(\tau|\theta)$ in termini della probabilità stessa e del gradiente del suo logaritmo.

$$\nabla_\theta J(\theta) = \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta) G_0(\tau) d\tau.$$

3. **Riformulare come aspettativa**: L'espressione risultante è ora l'aspettativa di un nuovo termine, il che ci permette di stimarla tramite campionamento.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[G_0(\tau)\nabla_\theta \log P(\tau|\theta)].$$

4. **Esplicitare la probabilità della traiettoria**: La probabilità di una traiettoria è il prodotto delle probabilità delle singole transizioni. Il suo logaritmo è una somma.

$$\log P(\tau|\theta) = \log p(S_0) + \sum_{t=0}^{T-1}(\log \pi_\theta(A_t|S_t) + \log p(S_{t+1}|S_t, A_t)).$$

5. **Calcolare il gradiente del logaritmo**: Quando deriviamo rispetto a θ, solo i termini che dipendono dalla politica (i termini $\log \pi_\theta$) sopravvivono.

$$\nabla_\theta \log P(\tau|\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t|S_t).$$

6. **Sostituire e Semplificare**: Sostituendo questo risultato nell'aspettativa, otteniamo una forma del gradiente che può essere stimata da traiettorie campionate.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[G_0(\tau)\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(A_t|S_t)\right].$$

Questa stima ha alta varianza. Per ridurla, si introducono due importanti miglioramenti. Primo, si sfrutta la **causalità**: un'azione A_t compiuta al tempo t può influenzare solo le ricompense future (R_{t+k+1} per $k \geq 0$), non quelle passate. Pertanto, quando si valuta l'impatto di $\nabla_\theta \log \pi_\theta(A_t|S_t)$, è più corretto moltiplicarlo per il ritorno futuro G_t anziché per il ritorno totale G_0. Secondo, si introduce una **baseline** $b(S_t)$ che dipende solo dallo stato, ottenendo la forma basata sulla Advantage function, molto più stabile:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi\left[A_\pi(S_t, A_t)\nabla_\theta \log \pi_\theta(A_t|S_t)\right].$$

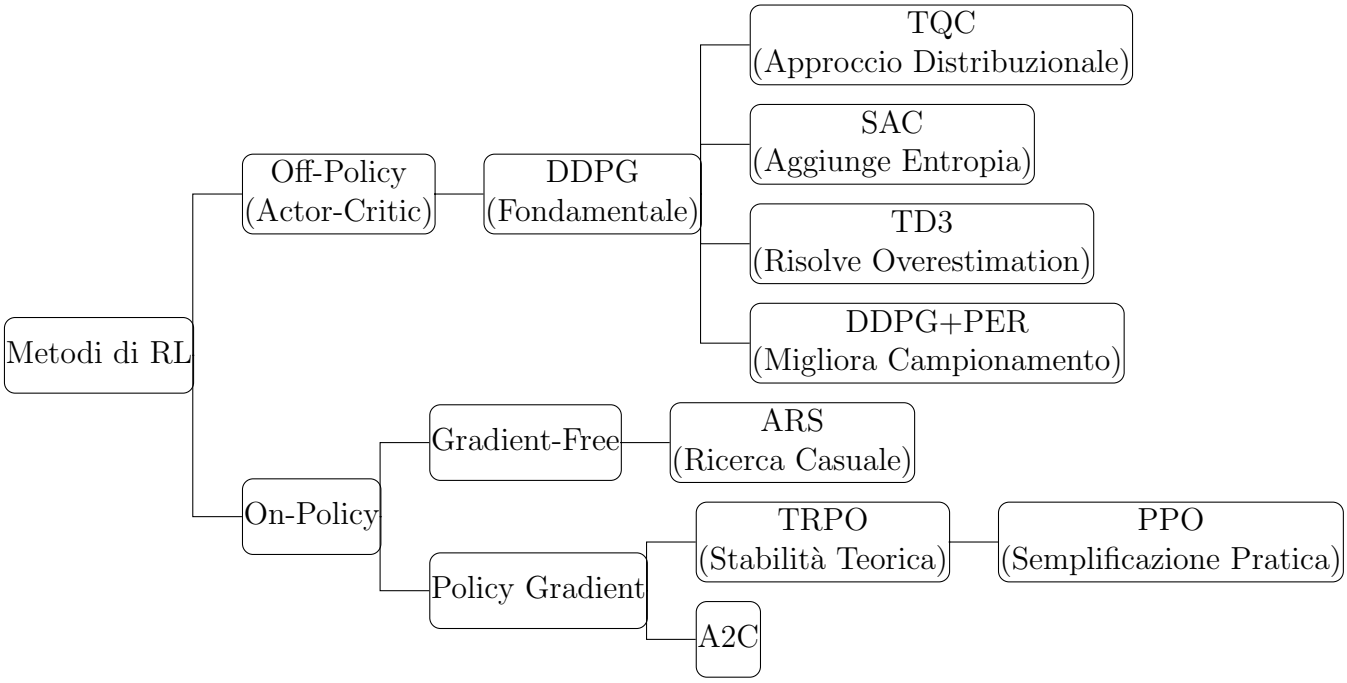
L'introduzione di una baseline $b(S_t)$ non introduce bias nella stima del gradiente. Questo è un risultato fondamentale che si dimostra mostrando che il valore atteso del termine

aggiunto è zero:

$$\begin{aligned}
\mathbb{E}_\pi\big[b(S_t)\,\nabla_\theta \log \pi_\theta(A_t|S_t)\big] &= \mathbb{E}_{S_t}\left[\sum_a \pi_\theta(a|S_t) b(S_t) \nabla_\theta \log \pi_\theta(a|S_t)\right] \\
&= \mathbb{E}_{S_t}\left[b(S_t)\sum_a \pi_\theta(a|S_t)\frac{\nabla_\theta \pi_\theta(a|S_t)}{\pi_\theta(a|S_t)}\right] \\
&= \mathbb{E}_{S_t}\left[b(S_t)\sum_a \nabla_\theta \pi_\theta(a|S_t)\right] \\
&= \mathbb{E}_{S_t}\left[b(S_t)\nabla_\theta \sum_a \pi_\theta(a|S_t)\right] \\
&= \mathbb{E}_{S_t}\left[b(S_t)\nabla_\theta 1\right] = 0.
\end{aligned}$$

2.1 Albero Genealogico degli Algoritmi

La seguente figura illustra le relazioni evolutive tra gli algoritmi che verranno discussi, evidenziando come le idee più recenti siano state costruite per superare i limiti dei loro predecessori.



3 Algoritmi Actor-Critic

3.1 La Motivazione: Superare i Limiti del Policy Gradient

Nella nostra analisi del Teorema del Policy Gradient, siamo giunti a una formulazione elegante e potente per il gradiente della funzione obiettivo:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[A_\pi(S_t, A_t) \nabla_\theta \log \pi_\theta(A_t|S_t) \right].$$

Questa equazione ci dice come aggiornare i parametri θ della nostra politica per migliorarla. Tuttavia, essa presenta una sfida pratica insormontabile: per calcolare l'aggiornamento, abbiamo bisogno di conoscere la vera *Advantage function* $A_\pi(s, a)$.
Calcolare $A_\pi(s, a) = q_\pi(s, a) - v_\pi(s)$ richiede la conoscenza delle vere *value functions* q_π e v_π, che sono esse stesse delle aspettative sul futuro e, in un contesto di apprendimento, sono ignote.
Ci troviamo di fronte a un problema circolare: per migliorare la politica, dobbiamo conoscere il suo valore; ma per conoscere il suo valore, dobbiamo seguire la politica e osservarne i risultati. Come possiamo risolvere questo dilemma? La risposta risiede nell'approssimazione. Invece di calcolare il valore esatto, lo stimiamo. Questo ci porta direttamente al cuore dell'architettura Actor-Critic.
L'idea fondamentale è quella di mantenere due strutture distinte che apprendono in parallelo: una per la politica e una per la stima del valore. Questa divisione dei compiti dà il nome alla famiglia di algoritmi:

- L'**Attore** (Actor), che corrisponde alla politica parametrizzata $\pi_\theta(a|s)$. Il suo ruolo è quello di agire, di scegliere le azioni da compiere nell'ambiente.
- Il **Critico** (Critic), che corrisponde a una stima della value function, parametrizzata da ϕ (es. $V_\phi(s) \approx v_\pi(s)$). Il suo ruolo è quello di osservare e valutare, o "criticare", le azioni compiute dall'attore, fornendo un segnale di feedback per l'apprendimento.

L'attore e il critico imparano l'uno dall'altro in un ciclo virtuoso: l'attore esplora l'ambiente, generando esperienze; il critico osserva queste esperienze per imparare a valutare meglio la politica dell'attore; l'attore, a sua volta, utilizza il feedback sempre più accurato del critico per migliorare la propria politica.

3.2 Il Critico: Apprendere la Funzione Valore tramite Temporal Difference

Il compito del critico è apprendere una stima della state-value function $v_\pi(s)$ per la politica corrente π_θ. Poiché non abbiamo un modello dell'ambiente, non possiamo usare metodi di Programmazione Dinamica. Invece, il critico apprende dall'esperienza diretta, utilizzando metodi di **Temporal Difference (TD) learning**.
Dopo ogni transizione $(S_t, A_t, R_{t+1}, S_{t+1})$, il critico calcola il **TD error**, δ_t. Questo segnale è al centro di tutto l'apprendimento e rappresenta una misura della "sorpresa" del critico: la differenza tra la stima del ritorno che si aspettava e una stima migliore ottenuta dopo un passo nell'ambiente.

$$\delta_t = R_{t+1} + \gamma V_\phi(S_{t+1}) - V_\phi(S_t).$$

Il termine $R_{t+1} + \gamma V_\phi(S_{t+1})$ è chiamato *TD target*. È una stima del ritorno G_t che combina la ricompensa reale osservata, R_{t+1}, con la stima corrente del valore dello stato successivo,

$V_\phi(S_{t+1})$.
L'obiettivo del critico è rendere le sue stime il più accurate possibile, ovvero minimizzare il TD error nel tempo. Questo viene formalizzato come un problema di ottimizzazione. La funzione di perdita (loss function) per il critico è tipicamente l'errore quadratico medio:

$$L(\phi) = \tfrac{1}{2}\,\delta_t^2.$$

Per minimizzare questa perdita, il critico aggiorna i suoi parametri ϕ tramite discesa del gradiente:

$$\phi \leftarrow \phi - \alpha_C \nabla_\phi L(\phi) = \phi - \alpha_C\big(\delta_t \nabla_\phi \delta_t\big) \approx \phi + \alpha_C\,\delta_t\,\nabla_\phi V_\phi(S_t),$$

dove α_C è il learning rate del critico. L'ultimo passaggio è una **semi-gradient approximation**. Si chiama "semi-gradiente" perché si deriva la loss solo rispetto alla stima $V_\phi(S_t)$, ignorando la dipendenza del TD target da ϕ (attraverso il termine $V_\phi(S_{t+1})$). Sebbene non sia una vera discesa del gradiente sull'errore di Bellman, questa approssimazione è computazionalmente efficiente e ha dimostrato di convergere in modo stabile in pratica, costituendo la base della maggior parte degli algoritmi TD.

3.3 L'Attore: Migliorare la Politica sulla Base della Critica

L'attore aggiorna i parametri della sua politica θ seguendo il gradiente della performance. Come abbiamo visto, il gradiente può essere stimato usando una stima della Advantage function, $\hat{A}_t$. L'intuizione geniale degli algoritmi Actor-Critic è che **il TD error δ_t calcolato dal critico è una stima (distorta ma a bassa varianza) della Advantage function**.
Perché? Ricordiamo che $A_\pi(S_t, A_t) = q_\pi(S_t, A_t) - v_\pi(S_t)$. Inoltre, $q_\pi(S_t, A_t) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t, A_t]$. Sostituendo le nostre stime V_ϕ al posto dei valori veri, otteniamo:

$$\hat{A}_t \approx \mathbb{E}_\pi[R_{t+1} + \gamma V_\phi(S_{t+1})|S_t, A_t] - V_\phi(S_t).$$

Il TD error $\delta_t = R_{t+1} + \gamma V_\phi(S_{t+1}) - V_\phi(S_t)$ è una versione campionata (single-sample) di questa stima.
L'attore può quindi utilizzare direttamente δ_t per aggiornare i suoi parametri. L'aggiornamento per θ diventa un passo nella direzione del gradiente stimato:

$$\theta \leftarrow \theta + \alpha_A \delta_t \nabla_\theta \log \pi_\theta(A_t|S_t),$$

dove α_A è il learning rate dell'attore. Se $\delta_t > 0$, l'aggiornamento aumenta la probabilità logaritmica dell'azione A_t, rendendola più probabile in futuro. Se $\delta_t < 0$, la probabilità di A_t viene ridotta.

3.4 Il Processo di Apprendimento Complessivo e A2C/A3C

Mettendo insieme i pezzi, il ciclo di apprendimento di un algoritmo Actor-Critic si svolge come segue ad ogni passo t:

1. L'attore osserva lo stato S_t e seleziona un'azione $A_t \sim \pi_\theta(\cdot|S_t)$.

2. L'agente esegue l'azione A_t e osserva la ricompensa R_{t+1} e il nuovo stato S_{t+1}.

3. Il critico calcola il TD error: $\delta_t = R_{t+1} + \gamma V_\phi(S_{t+1}) - V_\phi(S_t)$.
4. Il critico aggiorna i suoi parametri: $\phi \leftarrow \phi + \alpha_C \delta_t \nabla_\phi V_\phi(S_t)$.
5. L'attore aggiorna i suoi parametri: $\theta \leftarrow \theta + \alpha_A \delta_t \nabla_\theta \log \pi_\theta(A_t|S_t)$.

Questa formulazione è la base dell'algoritmo **Advantage Actor-Critic (A2C)**. Una sua importante estensione pratica è **Asynchronous Advantage Actor-Critic (A3C)** [10], che utilizza worker paralleli per interagire con copie multiple dell'ambiente. I worker aggiornano una versione globale dei parametri θ e ϕ in modo asincrono. Questo parallelismo ha un effetto stabilizzante sull'apprendimento, poiché le esperienze raccolte dai diversi worker sono decorrelate, rompendo le dipendenze temporali che possono rendere instabile l'apprendimento online.

4 Metodi On-Policy Avanzati e il Concetto di Stabilità

4.1 Il Problema Fondamentale: l'Instabilità degli Aggiornamenti di Policy

I metodi Policy Gradient, nella loro forma più semplice, aggiornano i parametri della politica θ seguendo la direzione del gradiente stimato, $\nabla_\theta J(\theta)$. Sebbene questo approccio sia teoricamente fondato, in pratica soffre di una notevole fragilità: la scelta della dimensione del passo di aggiornamento (o *learning rate*) è critica. Un passo troppo piccolo porta a una convergenza estremamente lenta, mentre un passo troppo grande può causare un collasso catastrofico della performance. Un singolo aggiornamento errato può spostare la politica in una regione pessima dello spazio dei parametri, dalla quale potrebbe non riprendersi mai.
Questa sensibilità deriva dal fatto che l'aggiornamento viene effettuato nello spazio dei parametri, ma il suo effetto nello spazio delle politiche (cioè sul comportamento effettivo dell'agente) è altamente non lineare e difficile da prevedere. La necessità di garantire aggiornamenti stabili e monotoni, o quasi, ha portato allo sviluppo di metodi più avanzati.

4.2 Trust Region Policy Optimization (TRPO)

Trust Region Policy Optimization (TRPO) [3] è stato il primo algoritmo a formalizzare rigorosamente l'idea di controllare la dimensione dell'aggiornamento della politica. Invece di limitare la magnitudine del vettore dei parametri θ, TRPO limita direttamente il "cambiamento comportamentale" della politica. L'idea centrale è di massimizzare il miglioramento della politica ad ogni passo, con il vincolo che la nuova politica rimanga all'interno di una "regione di fiducia" (*trust region*) attorno alla vecchia politica.

4.2.1 L'Obiettivo Surrogato e il Vincolo sulla Politica

TRPO formalizza il problema di ottimizzazione come la massimizzazione di una funzione obiettivo "surrogata" $L(\theta)$, che approssima il miglioramento atteso della performance, soggetta a un vincolo. L'obiettivo è:

$$L_{\pi_{\theta_{\text{old}}}}(\theta) = \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}, a\sim\pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \hat{A}_{\theta_{\text{old}}}(s,a) \right],$$

dove $\rho_{\theta_{\text{old}}}$ è la distribuzione degli stati visitati dalla vecchia politica $\pi_{\theta_{\text{old}}}$, e $\hat{A}$ è una stima della *Advantage function*. Questo obiettivo misura il miglioramento atteso della nuova politica π_θ valutato usando i dati raccolti con la vecchia politica.
Per garantire la stabilità, questo miglioramento viene massimizzato sotto un vincolo che limita la "distanza" tra la vecchia e la nuova politica. Questa distanza è misurata tramite la **Divergenza di Kullback-Leibler (KL)**. La divergenza KL tra due distribuzioni di probabilità p e q è definita come:

$$D_{KL}(p\|q) = \sum_x p(x) \log \frac{p(x)}{q(x)}.$$

Intuitivamente, essa misura l'informazione persa quando si usa la distribuzione q per approssimare la distribuzione p. È una misura non simmetrica ($D_{KL}(p\|q) \neq D_{KL}(q\|p)$) e non negativa, che vale zero solo se p e q sono identiche. Il problema di ottimizzazione di TRPO è quindi:

$$\begin{aligned} \underset{\theta}{\text{massimizzare}} \quad & L_{\pi_{\theta_{\text{old}}}}(\theta) \\ \text{soggetto a} \quad & \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}[D_{KL}(\pi_{\theta_{\text{old}}}(\cdot|s)\|\pi_\theta(\cdot|s))] \leq \delta. \end{aligned}$$

Il parametro δ definisce la dimensione della "regione di fiducia".

4.2.2 Soluzione Pratica e la Matrice di Informazione di Fisher (FIM)

Risolvere questo problema di ottimizzazione vincolata è computazionalmente costoso. TRPO lo affronta approssimando l'obiettivo al primo ordine e il vincolo al secondo ordine tramite uno sviluppo di Taylor attorno a θ_{old}. L'approssimazione quadratica del vincolo KL introduce un concetto fondamentale dalla teoria dell'informazione: la **Matrice di Informazione di Fisher (Fisher Information Matrix, FIM)**.

- **Intuizione e Ruolo**: La FIM può essere interpretata come una misura della curvatura locale dello spazio delle distribuzioni di probabilità. Dal punto di vista statistico, la FIM è il valore atteso dell'Hessiano del log-likelihood negativo e quantifica l'informazione che una variabile casuale osservabile porta con sé riguardo ai parametri sconosciuti di una distribuzione che la modella. In questo contesto, essa definisce una metrica nello spazio dei parametri che ci dice quanto un piccolo cambiamento nei parametri θ influenzi il comportamento della politica π_θ. Usare la FIM permette di fare passi di aggiornamento che hanno una dimensione costante in termini di cambiamento della politica, piuttosto che in termini di cambiamento dei parametri.

- **Definizione Matematica**: La FIM, F, è una matrice definita come il valore atteso del prodotto esterno del gradiente del logaritmo della verosimiglianza della politica:

$$F = \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}, a\sim\pi_{\theta_{\text{old}}}}\Big[\nabla_\theta \log \pi_\theta(a|s)|_{\theta=\theta_{\text{old}}} \; \nabla_\theta \log \pi_\theta(a|s)|^\top_{\theta=\theta_{\text{old}}}\Big].$$

- **Applicazione in TRPO**: Con la FIM, il vincolo KL viene approssimato come:

$$\mathbb{E}_s[D_{KL}(\pi_{\theta_{\text{old}}}\|\pi_\theta)] \approx \frac{1}{2}(\theta - \theta_{\text{old}})^T F (\theta - \theta_{\text{old}}).$$

  Questo trasforma il problema di ottimizzazione in un problema quadratico vincolato, che TRPO risolve in modo efficiente (senza costruire esplicitamente la matrice F) tramite il metodo del gradiente coniugato.

4.3 Proximal Policy Optimization (PPO)

Sebbene TRPO sia robusto e teoricamente ben fondato, la sua implementazione è complessa a causa della necessità di un solutore di secondo ordine. **Proximal Policy Optimization (PPO)** [4] nasce con l'obiettivo di ottenere benefici simili a TRPO (stabilità e affidabilità) utilizzando solo ottimizzazione di primo ordine, rendendolo molto più semplice da implementare e compatibile con architetture complesse.
L'idea chiave di PPO non è imporre un vincolo rigido, ma modificare la funzione obiettivo per disincentivare aggiornamenti che cambiano troppo la politica. Questo viene fatto tramite un meccanismo di "clipping".

4.3.1 L'Obiettivo Surrogato Clippato

PPO parte, come TRPO, dal rapporto di probabilità tra la nuova e la vecchia politica:

$$r_t(\theta) = \frac{\pi_\theta(A_t|S_t)}{\pi_{\theta_{\text{old}}}(A_t|S_t)}.$$

Questo rapporto indica quanto è più (o meno) probabile l'azione A_t sotto la nuova politica. Se $r_t(\theta) > 1$, l'azione è diventata più probabile. Se $r_t(\theta) < 1$, è diventata meno probabile. L'obiettivo standard del policy gradient cercherebbe di spingere $r_t(\theta)$ all'infinito se il vantaggio $\hat{A}_t$ è positivo. PPO previene questo comportamento introducendo una versione "clippata" del rapporto:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\big(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\big)\Big].$$

Analizziamo questa espressione:

- Il termine $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ confina il valore del rapporto di probabilità all'interno dell'intervallo $[1-\epsilon, 1+\epsilon]$.
- La funzione min seleziona il valore più basso tra l'obiettivo originale e l'obiettivo clippato.

L'effetto di questa formulazione dipende dal segno del vantaggio:

- **Se $\hat{A}_t > 0$ (l'azione è stata migliore della media)**: L'agente vorrebbe aumentare la probabilità di questa azione, ovvero aumentare $r_t(\theta)$. Tuttavia, la funzione min e il clipping a $1+\epsilon$ pongono un tetto al miglioramento. L'obiettivo smette di crescere una volta che l'aggiornamento della politica ha reso l'azione $1+\epsilon$ volte più probabile. Questo previene un passo troppo avido.
- **Se $\hat{A}_t < 0$ (l'azione è stata peggiore della media)**: L'agente vorrebbe diminuire la probabilità di questa azione, ovvero diminuire $r_t(\theta)$. La funzione min e il clipping a $1-\epsilon$ pongono un limite a quanto la probabilità possa essere ridotta in un singolo aggiornamento.

In sostanza, PPO crea una "regione di fiducia" soft, penalizzando gli aggiornamenti che si allontanano troppo dalla politica precedente, ma lo fa tramite una semplice modifica alla funzione obiettivo, rendendolo compatibile con qualsiasi ottimizzatore basato su gradiente.

5 Metodi Off-Policy: Apprendimento da Dati Esterni

5.1 La Distinzione Fondamentale: On-Policy vs. Off-Policy

Una delle distinzioni più importanti nella classificazione degli algoritmi di Reinforcement Learning riguarda la fonte dei dati utilizzati per l'apprendimento e la politica che si intende migliorare. Questa distinzione porta a due famiglie di metodi con filosofie e implicazioni pratiche molto diverse.

- **Apprendimento On-Policy**: In questa categoria di metodi, l'agente apprende il valore della politica che sta attualmente eseguendo. Le azioni vengono scelte secondo una politica π, e i dati raccolti (le transizioni $(S_t, A_t, R_{t+1}, S_{t+1})$) vengono utilizzati per valutare e migliorare quella stessa politica π. Questo approccio è concettualmente diretto: "impara facendo e migliora ciò che fai". Gli algoritmi visti finora, come A2C, TRPO e PPO, sono intrinsecamente on-policy. Il loro principale svantaggio è l'inefficienza dal punto di vista dei dati (*sample inefficiency*). Poiché la politica viene aggiornata continuamente, le esperienze raccolte con una politica "vecchia" diventano rapidamente obsolete e devono essere scartate. L'agente deve costantemente generare nuove esperienze con la sua politica più recente, un processo che può essere lento e costoso.

- **Apprendimento Off-Policy**: In questo paradigma, l'agente apprende il valore di una politica, detta *target policy* (π), utilizzando dati generati da un'altra politica, detta *behavior policy* (μ). Le due politiche sono disaccoppiate. Questo disaccoppiamento è estremamente potente. Tipicamente, la *target policy* è la politica che si desidera ottimizzare, e spesso è una politica deterministica e greedy rispetto alla stima corrente dei valori (la politica "migliore" che conosciamo). La *behavior policy*, invece, è la politica effettivamente eseguita dall'agente per raccogliere dati; essa è tipicamente stocastica e più esplorativa (es. una politica ϵ-greedy) per garantire che un'ampia varietà di stati e azioni venga visitata. Il vantaggio cruciale è che questo approccio permette di riutilizzare le esperienze passate, che possono essere memorizzate in una grande memoria chiamata **replay buffer**.

Il **replay buffer** è una struttura dati, tipicamente una coda FIFO (First-In, First-Out) di dimensione finita, che archivia le transizioni (s, a, r, s') man mano che vengono raccolte. Durante l'addestramento, invece di usare solo l'ultima transizione, l'algoritmo campiona un mini-batch di transizioni casualmente dal buffer. Questo meccanismo ha due benefici fondamentali:

1. **Migliora la stabilità dell'apprendimento**: Le esperienze sequenziali sono altamente correlate. Campionare casualmente dal buffer rompe queste correlazioni temporali, rendendo i dati più simili a campioni indipendenti e identicamente distribuiti (IID), un'assunzione su cui si basano molti algoritmi di ottimizzazione come la discesa del gradiente stocastico.

2. **Aumenta l'efficienza dei dati**: Ogni transizione può essere riutilizzata più volte per gli aggiornamenti dei parametri, permettendo all'agente di estrarre più informazione da ogni interazione con l'ambiente.

5.2 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) [5] rappresenta un punto di riferimento per gli algoritmi off-policy, estendendo con successo le idee di Deep Q-Networks (DQN) al dominio delle azioni continue. È un algoritmo di tipo Actor-Critic che apprende una politica deterministica.

5.2.1 Il Critico: Apprendimento della Action-Value Function

Poiché siamo in un contesto off-policy, non è sufficiente apprendere una state-value function. Dobbiamo stimare $q_*(s, a)$, il valore di compiere un'azione arbitraria a in uno stato s e poi seguire la politica ottimale. Il critico, $Q_\phi(s, a)$, è una rete neurale che approssima questa funzione. Viene addestrato minimizzando l'errore di Bellman su mini-batch di transizioni campionate dal replay buffer $\mathcal{D}$. La funzione di perdita (loss) è:

$$L(\phi) = \mathbb{E}_{(s_t,a_t,r_{t+1},s_{t+1})\sim\mathcal{D}} \left[(y_t - Q_\phi(s_t, a_t))^2 \right].$$

Il target y_t è calcolato utilizzando le reti *target*, una copia delle reti dell'attore e del critico i cui pesi (θ', ϕ') vengono aggiornati lentamente. Questo previene l'instabilità che deriverebbe dall'usare una rete che cambia ad ogni passo per calcolare il proprio target. Il target è:

$$y_t = r_{t+1} + \gamma Q_{\phi'}(s_{t+1}, \mu_{\theta'}(s_{t+1})).$$

Qui, l'azione successiva è scelta dalla *target policy* dell'attore, $\mu_{\theta'}$, e il suo valore è calcolato dal *target critic*, $Q_{\phi'}$.

5.2.2 L'Attore: Apprendimento della Politica Deterministica

L'attore, $\mu_\theta(s)$, è una rete neurale che mappa uno stato s a un'azione specifica a. Il suo obiettivo è produrre azioni che massimizzino il valore previsto dal critico. Pertanto, la sua funzione obiettivo è massimizzare il valore Q atteso:

$$J(\theta) = \mathbb{E}_{s_t\sim\mathcal{D}}[Q_\phi(s_t, \mu_\theta(s_t))].$$

L'attore aggiorna i suoi parametri θ tramite discesa del gradiente su questa funzione obiettivo. Applicando la regola della catena, si ottiene il gradiente deterministico di policy:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t\sim\mathcal{D}}[\nabla_a Q_\phi(s, a)|_{s=s_t,a=\mu_\theta(s_t)} \nabla_\theta \mu_\theta(s)|_{s=s_t}].$$

Questo gradiente "dice" all'attore come modificare i suoi parametri per produrre azioni che portino a un valore Q più alto, secondo la stima attuale del critico.

5.3 Twin Delayed DDPG (TD3)

Il DDPG, sebbene potente, soffre di una tendenza a sovrastimare sistematicamente i valori Q. Questo **overestimation bias** è un problema noto negli algoritmi basati su Q-learning. Emerge perché l'operatore di massimo nell'equazione di Bellman, usato per calcolare il valore target, tende a selezionare preferenzialmente azioni i cui valori stimati sono casualmente alti a causa del rumore di approssimazione della rete neurale. Questo errore di sovrastima può poi propagarsi all'indietro durante l'addestramento, portando a stime di valore gonfiate e a politiche sub-ottimali. **Twin Delayed DDPG (TD3)** [7] introduce tre modifiche cruciali per mitigare questo problema.

1. **Clipped Double Q-Learning**: Per contrastare la sovrastima, TD3 apprende due reti critiche, Q_{ϕ_1} e Q_{ϕ_2}. Quando si calcola il target y_t, si usa il minimo dei valori predetti dalle due reti target. Questo approccio conservativo riduce la probabilità di propagare sovrastime. Il target diventa:

$$y_t = r_{t+1} + \gamma \min_{i=1,2} Q_{\phi'_i}(s_{t+1}, \tilde{a}_{t+1}).$$

2. **Target Policy Smoothing**: Per rendere la stima del valore più robusta a errori di approssimazione, TD3 aggiunge un piccolo rumore clippato all'azione scelta dalla politica target prima di calcolare il valore Q nel target. Questo regolarizza l'apprendimento, "appiattendo" la funzione valore.

$$\tilde{a}_{t+1} = \text{clip}\left(\mu_{\theta'}(s_{t+1}) + \epsilon,\ a_{\text{low}}, a_{\text{high}}\right), \qquad \epsilon \sim \text{clip}\left(\mathcal{N}(0, \sigma^2), -c, c\right).$$

3. **Delayed Policy Updates**: Per aumentare la stabilità, l'attore e le reti target vengono aggiornate meno frequentemente del critico (es. ogni due aggiornamenti del critico). Questo permette al critico di convergere verso una stima del valore più accurata prima che la politica venga modificata sulla base di essa.

5.4 Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) [8] è un algoritmo off-policy che riformula l'obiettivo del RL in un framework di **massima entropia**. Invece di massimizzare solo il ritorno, l'agente cerca di massimizzare una combinazione di ritorno ed entropia della politica.

5.4.1 Il Framework della Massima Entropia

L'entropia di una distribuzione di probabilità è una misura della sua incertezza o casualità. Per una politica stocastica $\pi(\cdot|s)$, l'entropia è definita come:

$$\mathcal{H}(\pi(\cdot|s)) = \mathbb{E}_{a\sim\pi(\cdot|s)}[-\log \pi(a|s)].$$

Una politica ad alta entropia è più "casuale", mentre una a bassa entropia è più deterministica. L'obiettivo di SAC è trovare una politica che massimizzi il seguente obiettivo:

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t (R_{t+1} + \alpha \mathcal{H}(\pi(\cdot|S_t)))\right].$$

Il "coefficiente di temperatura" α bilancia l'importanza relativa tra la massimizzazione della ricompensa e la massimizzazione dell'entropia. L'inclusione dell'entropia nell'obiettivo serve a due scopi cruciali:

- **Incoraggia l'esplorazione**: Premiando la politica per essere il più stocastica possibile (entro i limiti del compito), l'agente è incentivato a esplorare un'ampia gamma di azioni promettenti, piuttosto che convergere prematuramente a una singola azione deterministica che potrebbe essere solo un ottimo locale.

- **Migliora la robustezza**: Una politica ad alta entropia è meno suscettibile a piccoli cambiamenti nell'ambiente e può adattarsi più facilmente. Impara a "tenere aperte le opzioni", il che può portare a un apprendimento più stabile e a soluzioni finali più robuste.

L'implementazione moderna di SAC utilizza due critici (come TD3) e apprende automaticamente il parametro α tramite un'ulteriore discesa del gradiente per bilanciare dinamicamente il trade-off tra ricompensa ed entropia.

6 Estensioni e Algoritmi Alternativi

In questa sezione, analizziamo le formulazioni matematiche di tre approcci aggiuntivi: un metodo gradient-free (ARS), una modifica al campionamento off-policy (PER) e un algoritmo Actor-Critic avanzato per la gestione del bias (TQC).

6.1 Augmented Random Search (ARS)

Augmented Random Search (ARS) [9] è un algoritmo on-policy e *gradient-free* che ottimizza la politica operando direttamente nello spazio dei parametri θ. Invece di calcolare il gradiente, esplora direzioni casuali e aggiorna i parametri in base alla performance osservata.

L'algoritmo procede come segue:

1. **Campionamento delle perturbazioni**: Ad ogni iterazione, si campionano N vettori di perturbazione casuale $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_N$ da una distribuzione isotropa, tipicamente una gaussiana $\mathcal{N}(0, \mathbf{I})$.

2. **Valutazione della politica**: Per ogni $\boldsymbol{\delta}_i$, si valuta la politica in due direzioni simmetriche, $\theta + \nu\boldsymbol{\delta}_i$ e $\theta - \nu\boldsymbol{\delta}_i$, dove $\nu > 0$ è un iperparametro che controlla la magnitudine del rumore. Si raccolgono i ritorni corrispondenti, $G(\theta + \nu\boldsymbol{\delta}_i)$ e $G(\theta - \nu\boldsymbol{\delta}_i)$.

3. **Aggiornamento dei parametri**: I parametri della politica vengono aggiornati seguendo una stima della direzione di massima pendenza, costruita come media pesata dei vettori di perturbazione. Il peso di ogni $\boldsymbol{\delta}_i$ è proporzionale alla differenza dei ritorni ottenuti. L'aggiornamento è:

$$\theta_{k+1} \leftarrow \theta_k + \frac{\alpha}{N\sigma_G} \sum_{i=1}^{N} \left[G(\theta_k + \nu\boldsymbol{\delta}_i) - G(\theta_k - \nu\boldsymbol{\delta}_i) \right] \boldsymbol{\delta}_i,$$

   dove α è il learning rate e σ_G è la deviazione standard di tutti i $2N$ ritorni raccolti, utilizzata per normalizzare l'aggiornamento e renderlo più stabile.

6.2 DDPG con Prioritized Experience Replay (PER)

Questa non è un nuovo algoritmo, ma una modifica cruciale al meccanismo di campionamento del DDPG standard, introdotta da Schaul et al. (2015) [6]. L'idea è di campionare le transizioni dal replay buffer $\mathcal{D}$ non in modo uniforme, ma con una probabilità proporzionale alla loro "importanza", misurata dall'errore di stima.

1. **Calcolo della priorità**: La priorità di una transizione i è basata sul valore assoluto del suo TD error, calcolato dal critico:

$$p_i = |\delta_i| + \epsilon = |y_i - Q_\phi(s_i, a_i)| + \epsilon,$$

   dove ϵ è una piccola costante positiva che assicura che nessuna transizione abbia probabilità zero.

2. **Probabilità di campionamento**: La probabilità di campionare la transizione i è data da:
$$P(i) = \frac{p_i^\beta}{\sum_{k \in \mathcal{D}} p_k^\beta},$$
dove l'esponente $\beta \in [0, 1]$ controlla il grado di prioritizzazione. Con $\beta = 0$ si ritorna al campionamento uniforme.

3. **Correzione del bias con Importance Sampling (IS)**: Il campionamento prioritario introduce un bias, poiché le transizioni con alto errore vengono viste più spesso. Per correggerlo, si pondera l'aggiornamento della loss di ogni transizione con un peso di *importance sampling* (IS):
$$w_i = \left(\frac{1}{|\mathcal{D}| \cdot P(i)} \right)^\eta,$$
dove $|\mathcal{D}|$ è la dimensione del buffer e $\eta \in [0, 1]$ è un esponente che bilancia la correzione. I pesi vengono normalizzati dividendoli per $\max_j w_j$.

4. **Funzione di perdita ponderata**: La loss del critico del DDPG viene modificata per includere questi pesi, dando più importanza agli errori delle transizioni prioritarie durante la correzione del bias:
$$L(\phi) = \mathbb{E}_{i \sim P} \left[w_i \cdot (y_i - Q_\phi(s_i, a_i))^2 \right].$$

6.3 Truncated Quantile Critics (TQC)

Truncated Quantile Critics (TQC) [15] è un algoritmo off-policy di tipo Actor-Critic che affronta il bias di sovrastima in modo più fondamentale rispetto a TD3. Invece di stimare il valore atteso del ritorno, apprende la sua intera **distribuzione cumulativa** tramite la regressione a quantili.

1. **Apprendimento distribuzionale**: TQC utilizza un insieme di N reti critiche, $\{Q_{\phi_i}(s, a)\}_{i=1}^N$. Ciascun critico è addestrato a stimare un diverso quantile della distribuzione dei ritorni. Questo viene fatto minimizzando la **quantile Huber loss**.

2. **Calcolo del target distribuzionale**: Per calcolare il valore target, si procede in più passi:
   - Si campiona un'azione dal target actor per lo stato successivo: $\tilde{a}_{t+1} \sim \pi_{\theta'}(\cdot|s_{t+1})$.
   - Si ottengono N stime del Q-value per lo stato successivo, una da ciascuna delle N reti critiche target: $\{Q_{\phi'_j}(s_{t+1}, \tilde{a}_{t+1})\}_{j=1}^N$.

3. **Truncation**: Questa è l'idea chiave di TQC. Per combattere la sovrastima, l'algoritmo scarta le k stime di Q-value più grandi tra le N ottenute. Questo "troncamento" rimuove le stime più ottimistiche, che sono la fonte principale del bias.

4. **Aggiornamento del critico**: Il valore target per l'aggiornamento dei critici è calcolato come la media delle $N - k$ stime di Q-value rimanenti (le più piccole):
$$y_t = r_{t+1} + \gamma \left(\frac{1}{N-k} \sum_{j=1}^{N-k} Q_{\phi'_j}(s_{t+1}, \tilde{a}_{t+1}) \right),$$

dove la somma è calcolata sulle stime ordinate. Questo target viene poi utilizzato all'interno della quantile loss per aggiornare i parametri ϕ di tutte le N reti critiche.

5. **Aggiornamento dell'attore**: L'attore viene aggiornato per massimizzare il valore atteso del ritorno, che viene approssimato come la media delle stime del primo insieme di critici (non target):

$$\max_{\theta} \mathbb{E}_{s\sim\mathcal{D}} \left[\frac{1}{N} \sum_{i=1}^{N} Q_{\phi_i}(s, \pi_\theta(\cdot|s)) \right].$$

7 Sintesi Comparativa e Classificazione degli Algoritmi

Basandoci sull'analisi rigorosa e intuitiva, possiamo classificare gli algoritmi discussi in base al loro profilo di performance atteso in un compito complesso come la massimizzazione del profitto V2G.

7.1 Classificazione delle Performance Attese

7.1.1 Performance Affidabile e allo Stato dell'Arte

Algoritmi che dovrebbero comportarsi meglio in modo consistente, combinando alta efficienza dei dati con stabilità.

- **TQC e SAC**: Rappresentano lo stato dell'arte per il controllo continuo off-policy. Entrambi affrontano il problema critico del *overestimation bias* in modi sofisticati (TQC tramite troncamento dei quantili, SAC tramite regolarizzazione entropica e double Q-learning). Ci aspettiamo che raggiungano il **profitto più elevato e più stabile**, trovando un equilibrio eccellente tra esplorazione e sfruttamento. La loro performance non è casuale, ma il risultato di meccanismi di stabilizzazione robusti.

7.1.2 Buona Performance ma con Compromessi (Complessità o Lentezza)

Algoritmi che possono raggiungere buoni risultati, ma a costo di una maggiore richiesta di dati, stabilità o complessità computazionale.

- **TD3**: È un algoritmo estremamente solido. Implementa meccanismi efficaci (Clipped Double Q-Learning, Delayed Updates) per mitigare i problemi del DDPG. Raggiungerà una performance molto alta e stabile, probabilmente appena al di sotto di TQC e SAC.

- **TRPO e PPO**: Sono i "maratoneti" del gruppo. La loro performance finale può essere buona, ma la loro natura on-policy li rende estremamente inefficienti in termini di dati. Richiederanno un **numero di passi di simulazione molto più elevato** per raggiungere un livello di profitto paragonabile a quello degli algoritmi off-policy. La loro forza è la stabilità (teorica per TRPO, pratica per PPO), non la velocità di apprendimento.

- **ARS**: Simile a PPO e TRPO, è molto inefficiente e richiede un'enorme quantità di interazioni per ottimizzare la politica tramite ricerca casuale. La sua applicabilità pratica in un problema V2G complesso è limitata dalla sua lentezza.

7.1.3 Apprendimento Rapido ma Meno Stabile

Algoritmi che imparano velocemente grazie all'efficienza dei dati, ma la cui performance finale è spesso inaffidabile a causa di problemi di stabilità intrinseci.

- **DDPG**: È l'archetipo di questa categoria. Essendo off-policy, apprende molto rapidamente nelle fasi iniziali. Tuttavia, il suo noto problema di *overestimation bias* lo rende incline a **convergere a politiche sub-ottimali o a divergere completamente**.

- **DDPG+PER**: L'aggiunta del Prioritized Experience Replay **accelera ulteriormente la fase di apprendimento** del DDPG. Tuttavia, non risolve il problema di stabilità di fondo e non garantisce una soluzione finale migliore o più stabile.

8 Verifica Empirica: Confronto con i Risultati della Simulazione

La validazione empirica è un passo cruciale per verificare se le proprietà teoriche degli algoritmi si traducono in performance reali. In questa sezione, confrontiamo le aspettative delineate in precedenza con i risultati ottenuti da quattro diversi scenari di simulazione V2G, concentrandoci esclusivamente sulla metrica del **Profitto Totale**.

8.1 Punti in Accordo con le Aspettative Teoriche

Dall'analisi dei risultati emergono diverse conferme delle debolezze e dei punti di forza teorici di alcuni algoritmi.

- **Instabilità del DDPG**: In quasi tutti gli scenari, il DDPG standard mostra una performance mediocre o addirittura fortemente negativa. Questo risultato è pienamente in linea con le aspettative: il suo noto problema di *overestimation bias* lo rende inaffidabile e incline a convergere verso politiche sub-ottimali che, in pratica, portano a una perdita economica.

- **Inefficienza di ARS**: L'algoritmo ARS si attesta costantemente tra i peggiori performer in termini di profitto. Questo conferma la sua natura *gradient-free* e la sua estrema inefficienza dei dati. In un problema complesso come il V2G, un approccio basato sulla ricerca casuale fatica a trovare una strategia profittevole senza un numero di interazioni proibitivo.

- **Limiti di DDPG+PER**: La variante con Prioritized Experience Replay non mostra un miglioramento decisivo rispetto al DDPG standard e in alcuni casi non riesce a generare profitto. Ciò supporta l'idea che, sebbene PER possa accelerare l'apprendimento, non risolve i problemi strutturali di stabilità del suo algoritmo di base.

8.2 Risultati Inattesi e Spunti di Riflessione

I risultati delle simulazioni presentano anche significative sorprese, sfidando la gerarchia di performance che la teoria suggerirebbe.

- **Performance Sorprendente degli Algoritmi On-Policy (PPO e A2C)**: Il risultato più inatteso è l'eccellente performance di PPO e A2C. Contrariamente alla previsione che la loro inefficienza dei dati li avrebbe penalizzati, questi algoritmi si classificano spesso tra i migliori, generando in alcuni scenari il profitto più alto tra tutti gli agenti di RL. Questo suggerisce che, per la specifica dinamica di questi scenari V2G, la **stabilità e la natura cauta degli aggiornamenti on-policy** potrebbero essere più importanti dell'efficienza dei dati. È possibile che gli algoritmi off-policy, più aggressivi, fatichino a gestire la volatilità del problema, mentre PPO e A2C convergono più lentamente verso soluzioni "sicure" ma altamente profittevoli.

- **Performance Inconsistente degli Algoritmi allo Stato dell'Arte (SAC e TQC)**: Contrariamente alle aspettative, SAC e TQC, considerati teoricamente superiori, mostrano una performance molto variabile. In alcuni scenari sono competitivi, ma in altri il loro profitto è mediocre o addirittura negativo. Questo risultato è un importante promemoria del fatto che la complessità di un algoritmo non garantisce una performance superiore. SAC e TQC introducono iperparametri aggiuntivi (il coefficiente di entropia α per SAC, il numero di critici e il livello di troncamento per TQC) che potrebbero richiedere un'attenta calibrazione. Una configurazione non ottimale potrebbe limitare la loro efficacia, portando a risultati inferiori rispetto ad algoritmi più semplici ma robusti come PPO.
- **Competitività dei Metodi Baseline**: È da notare come in alcuni scenari un metodo baseline (RR) riesca a superare la maggior parte, se non tutti, gli agenti di RL. Questo evidenzia che la complessità del Reinforcement Learning non è sempre necessaria e che una strategia euristica ben progettata può essere un avversario formidabile.

In conclusione, le simulazioni confermano le debolezze teoriche di algoritmi come DDPG e ARS, ma mettono fortemente in discussione la superiorità universale dei metodi off-policy più recenti. L'eccezionale performance degli algoritmi on-policy suggerisce che la stabilità dell'apprendimento è un fattore critico in questi scenari, a volte più decisivo della pura efficienza dei dati.

References

[1] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[2] Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems (NIPS) 12*.

[3] Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning (ICML)*.

[4] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[5] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[6] Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

[7] Fujimoto, S., van Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International conference on machine learning (ICML)*.

[8] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning (ICML)*.

[9] Mania, H., Guy, A., & Recht, B. (2018). Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in neural information processing systems (NeurIPS) 31*.

[10] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning (ICML)*.

[11] Sadeghi, A., et al. (2021). Deep reinforcement learning for frequency control in power grids with V2G. *IEEE Transactions on Smart Grid*, 12(6), 5338-5348.

[12] Su, Z., et al. (2020). Deep reinforcement learning-based scheduling for vehicle-to-grid services. *IEEE Transactions on Smart Grid*, 11(4), 3073-3084.

[13] Zou, Y., et al. (2021). A deep reinforcement learning approach for EV charging and V2G discharging scheduling. *Energy and AI*, 5, 100081.

[14] Logeshwaran, J., et al. (2022). A comparative study of deep reinforcement learning algorithms for energy management in commercial buildings. *Energy and Buildings*, 254, 111589

[15] Kuznetsov, A., et al. (2020). Controlling overestimation bias with truncated quantile critics. In *International Conference on Machine Learning (ICML)*.