Politecnico di Torino

POLITECNICO DI TORINO

DIPARTIMENTO DI INGEGNERIA MECCANICA E AEROSPAZIALE

Corso di Laurea Magistrale - Ingegneria Aerospaziale

TRAJECTORY OPTIMISATION WITH REINFORCEMENT LEARNING ALGORITHM

Neural Monte Carlo Tree Search - MCTS

Relatore:
Prof. Casalino Lorenzo

Co. Relatore:
Dott. Forestieri Andrea

Studente:
Luigi Galasso

Matricola n° 305459

Anno Accademico: 2023/2024

Alla famiglia e agli amici

ABSTRACT

Nell'ambito della ricerca spaziale e dell'intelligenza artificiale, questa tesi magistrale esplora l'integrazione del *Monte Carlo Tree Search* (MCTS), un avanzato algoritmo decisionale, con le ultime innovazioni nel campo del *Machine Learning* (ML).

Attraverso l'applicazione di tecniche derivate dal *Reinforcement Learning* (RL), quali le simulazioni Monte Carlo e l'impiego del *Markov Decision Process* (MDP), questo studio affronta complesse sfide di navigazione e pianificazione di missioni spaziali con l'obiettivo di ottimizzare le traiettorie nello spazio interplanetario.

Il nucleo della ricerca è rappresentato dalla sviluppo di una metodologia per simulare e ottimizzare decisioni strategiche in scenari spaziali, esplorando come le reti neurali profonde possano migliorare l'efficacia di MCTS.

L'integrazione di queste tecnologie ha portato alla creazione di un framework avanzato, ispirato all'architettura di AlphaZero, in grado di gestire la complessità delle traiettorie spaziali con un'efficienza e una precisione computazionale notevole.

L'analisi sperimentale, fulcro dell'indagine di questo elaborato, impiega i modelli sviluppati per trovare soluzioni al problema presentato nella *Global Trajectory Optimisation Competition* (GTOC), attraverso simulazioni numeriche e l'ottimizzazione degli iperparametri dei rispettivi modelli. I risultati evidenziano un miglioramento significativo nelle decisioni di navigazione e aprono nuove prospettive per l'esplorazione spaziale.

In conclusione, il contributo di questa tesi sottolinea l'efficienza dell'integrazione tra MCTS e tecniche avanzate di *Deep Learning* nel dominio della meccanica orbitale, gettando le basi per future ricerche e offrendo soluzioni innovative per affrontare le sfide emergenti nell'esplorazione spaziale.

Parole chiave: *Monte Carlo Tree Search* (MCTS), Meccanica Orbitale, *Machine Learning* (ML), *Deep Learning* (DL), *Reinforcement Learning* (RL), *Global Trajectory Optimisation Competition* (GTOC).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

SUMMARY OF NOTATION

ABBREVIATIONS

AI	*Artificial Intelligence*
ADAM	*Adaptive Momentum Estimation*
ADP	*Asynchronous Dynamic Programming*
ATD	*Asteroid Transfer Device*
BCEL	*Binary Cross-Entropy Loss*
CNN	*Convolutional Neural Network*
CEL	*Cross-Entropy Loss*
CCEL	*Categorical Cross-Entropy Loss*
DL	*Deep Learning*
DNN	*Deep Neural Network*
DP	*Dynamic Programming*
GTOC	*Global Trajectory Optimisation Competition*
GPI	*Generalized Policy Iteration*
J	*Performance Index*
LEO	*Low Earth Orbit*
MC	*Monte Carlo*
MCTS	*Monte Carlo Tree Search*
MDP	*Markov Decision Process*
MEO	*Medium Earth Orbit*
ML	*Machine Learning*
MLP	*Multi-Layer Perceptron*
MJD	*Modified Julian Date*
MRP	*Markov Reward Process*
MSE	*Mean Squared Error*
NA	*Nodo Ascendente*
NAG	*Nesterov Accelerated Gradient*
ND	*Nodo Discendente*
NMCTS	*Neural Monte Carlo Tree Search*
NN	*Neural Network*
PUCT	*Predictor Upper Confidence Bound*
RAVE	*Rapid Action Value Estimation*
RL	*Reinforcement Learning*
RMS	*Root Mean Square*
RNN	*Recurrent Neural Network*
SGD	*Stochastic Gradient Descent*
SNN	*Shallow Neural Network*
SOI	*Sfera di Influenza*
SVM	*Support Vector Machines*
TD	*Temporal-Difference*
UCB	*Upper Confidence Bound*
UCT	*Upper Confidence Bound for Trees*

Politecnico di Torino

SYMBOLS NOTATION

— Meccanica Orbitale —

$\odot$	*Sole*
$\male$	*Marte*
$\oplus$	*Terra*
a	*Semiasse Maggiore*
e	*Eccentricità*
$\mathscr{E}$	*Energia Meccanica Specifica*
G	*Costante di Gravitazione Universale*
h	*Momento della Quantità di Moto Specifico*
i	*Inclinazione*
μ	*Parametro Gravitazionale*
Ω	*Longitudine del Nodo Ascendente*
ω	*Argomento del Periastro*
p	*Semilatus Rectum*
ν	*Anomalia Vera*
v_∞	*Eccesso Iperbolico di Velocità*

— Machine Learning & Deep Learning —

$f[x,\phi]$	*Neural Network*
ϕ	*Pesi*
θ	*Bias*
μ	*Media*
σ	*Varianza*
$\nabla f()$	*Gradiente di una Funzione*
α	*Learning Rate*
γ	*Parametro di Momentum*
$a[\cdot]$	*Funzione di Attivazione*
$L[\phi]$	*Loss Function*

Politecnico di Torino

$\doteq$	Uguaglianza tramite una definizione
$\simeq$	Approssimativamente uguale
$\propto$	Direttamente Proporzionale
$Pr\{X = x\}$	
ε	Probabilità che l'agente scelga un'azione random in una ε-greedy policy
α o β	Step-size parameters
γ	Discount-rate parameter
π	Policy
$\pi(s)$	Azione eseguita allo stato s sotto la policy π
$\pi(a\|s)$	Probabilità di eseguire un'azione allo stato s sotto la policy stocastica π
$\mathbb{E}[\cdot]$	Valore atteso di una variabile casuale sotto la policy π
$\mathbb{1}_i$	$\mathbb{1}_i \doteq 1$ se i è True, altrimenti $\mathbb{1}_i = 0$
a, a'	Azione generica
A_t	Azione al tempo t
$A(s)$	Set di tutte le azioni possibili nello stato s
b	*Branching Factor*
G	Return
G_t	Return ottenuto al tempo t
$p(s', r\|s, a)$	Probabilità di transizione allo stato s', ottenendo un reward r, dallo stato s ed eseguendo l'azione a
$p(s'\|s, a)$	Probabilità di transizione allo stato s' dallo stato s ed eseguendo l'azione a
$q_\pi(s, a)$	Valore dell'azione a allo stato s sotto la policy π
$q_*(s, a)$	Valore dell'azione a allo stato s sotto la policy ottima
r	Reward generico
R	set di tutti i possibili Reward
R_t	Reward al tempo t
$r(s, a)$	Expected reward dallo stato s ed eseguendo l'azione a
$r(s, a, s')$	Expected reward, sotto la transizione dallo stato s allo stato s' ed eseguendo l'azione a
s, s'	Stato generico
S	Set di tutti gli stati, escluso lo stato terminale
S^+	Set di tutti gli stati, incluso lo stato terminale
S_t	Stato al tempo t
t	Time step
$v_\pi(s)$	Valore dello stato s sotto la policy π
$v_*(s)$	Valore dello stato s sotto la policy ottima

Politecnico di Torino

MECCANICA ORBITALE

Politecnico di Torino

Politecnico di Torino

CHAPTER 1

MECCANICA ORBITALE

1.1 FONDAMENTI DELLA MECCANICA ORBITALE

La meccanica orbitale si occupa dello studio del movimento di oggetti come pianeti, satelliti, sia naturali che artificiali, e altri corpi minori, che si muovono seguendo orbite determinate dall'attrazione gravitazionale. Questo campo di studio è fondamentale non solo per la pura ricerca astronomica, ma assume un ruolo chiave nella progettazione di missioni spaziali, consentendo di calcolare traiettorie precise per satelliti e sonde spaziali.

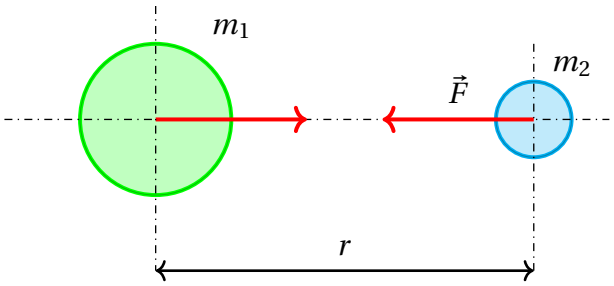


Figure 1.1: Forza esercitata su due corpi celesti secondo la legge di Gravitazione Universale.

L'impiego della legge di gravitazione universale formulata da Isaac Newton[1] ha segnato una svolta, permettendo lo sviluppo di sofisticati modelli matematici che descrivono con notevole precisione le orbite dei corpi celesti.

$$\vec{F} = G \cdot \frac{m_1 \cdot m_2}{r^2} \cdot \hat{r} \qquad G = 6.67430 \cdot 10^{-11} \quad \left[\frac{m^3}{kg \cdot s^2}\right] \tag{1.1}$$

In questa equazione, $\vec{F}$ denota la forza gravitazionale esercitata tra due masse, m_1 e m_2, a una distanza r l'una dall'altra, mentre $\hat{r}$ indica la direzione della forza. La costante di gravitazione universale, G, riflette l'intensità dell'interazione gravitazionale in tutto l'universo, elemento fondamentale nella predizione delle traiettorie orbitali.

Utilizzando l'equazione della gravitazione universale di Newton, e adottando un modello semplificato in cui i corpi celesti sono considerati come punti materiali, è possibile calcolare il moto dei corpi celesti sotto l'influenza della loro mutua attrazione gravitazionale. In questo modello, la forza esercitata su un corpo celeste è determinata dalla massa e dalla distanza rispetto agli altri corpi celesti, con un impatto maggiore esercitato dai corpi più massicci e più vicini.

Questo approccio, seppur semplificato, introduce il concetto del problema ad N-Corpi (*N-Body Problem*). La risoluzione del moto dei corpi risulta quindi essere un sistema complesso di equazioni differenziali che, a causa della loro interdipendenza[2], raramente ammette soluzioni analitiche esatte e necessita di essere affrontato attraverso metodi numerici.

Tuttavia, mediante un'opportuna approssimazione, possiamo considerare principalmente l'influenza del corpo celeste più massivo e vicino sull'oggetto in studio, riducendo il problema a un problema dei 2-Corpi (*2-Body Problem*).

[1] *Principia a Mathematica* (1687)
[2] Sistema di equazioni accoppiate.

Politecnico di Torino

Questa semplificazione consente di analizzare con precisione la traiettoria orbitale dell'oggetto, presupponendo che l'influenza predominante sia quella del corpo più significativo a livello gravitazionale.

L'accuratezza della soluzione del problema dei 2-Corpi è massima quando l'oggetto considerato orbita in prossimità del corpo celeste dominante, presupposto che sta alla base del concetto di *Sfere d'Influenza*. Queste ultime giocano un ruolo cruciale nella progettazione preliminare di missioni interplanetarie, permettendo di modellare il passaggio da un'area di predominante influenza gravitazionale di un corpo celeste (ad esempio, la Terra) a quella di un altro (come il Sole o un pianeta target).

Grazie all'approssimazione del problema dei 2-Corpi, è possibile calcolare analiticamente e in forma chiusa la traiettoria di una sonda spaziale o di un corpo celeste che orbita attorno a un corpo principale. Questo permette di prevedere la posizione futura dell'oggetto orbitante e di conoscere con precisione i parametri orbitali in funzione del tempo. L'approccio semplificato del problema dei 2-Corpi non solo facilita la comprensione e la progettazione delle traiettorie spaziali ma consente anche di dimostrare e applicare le leggi di Keplero, che descrivono le orbite dei pianeti nel sistema solare con notevole accuratezza.

1.1.1 Le Leggi del Moto di Keplero

Le leggi del moto di Keplero rappresentano una pietra miliare nella comprensione dei movimenti dei corpi celesti nel sistema solare. Formulate all'inizio del XVII secolo[3] da Johann Kepler, queste leggi descrivono il moto dei pianeti attorno al Sole, basandosi sull'accurata raccolta di dati astronomici effettuata da Tycho Brahe. Le leggi di Keplero furono rivoluzionarie perché fornirono una descrizione empirica e quantitativa delle orbite planetarie, preparando il terreno per la legge di gravitazione universale di Newton.

1. **Prima Legge di Keplero: Legge delle Orbite Ellittiche - 1609**:

   La prima legge di Keplero, conosciuta come la legge delle orbite ellittiche, afferma che ogni pianeta si muove lungo un'orbita ellittica, con il Sole situato in uno dei due fuochi dell'ellisse. Questo contraddistingue il movimento planetario dal concetto pre-kepleriano di orbite circolari perfette, introducendo una visione più precisa e matematicamente valida del sistema solare.

2. **Seconda Legge di Keplero: Legge delle Aree - 1609**:

   La seconda legge, nota come legge delle aree, stabilisce che una linea che collega un pianeta al Sole spazza aree uguali in tempi uguali. Questo implica che la velocità orbitale di un pianeta varia in modo tale che si muova più velocemente quando è vicino al Sole (*Perielio*) e più lentamente quando è lontano dal Sole (*Afelio*).

3. **Terza Legge di Keplero: Legge dei Periodi - 1619**:

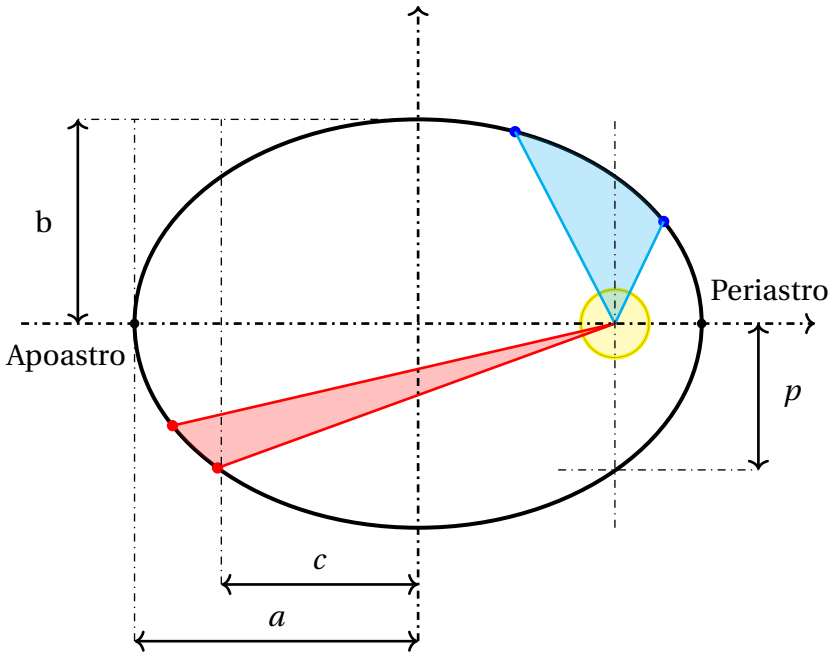
   La terza legge, spesso chiamata legge dei periodi, stabilisce che il quadrato del periodo orbitale di un pianeta è proporzionale al cubo del semi-maggiore asse della sua orbita. Matematicamente, questo può essere espresso come:

$$T^2 \propto a^3 \qquad T = 2\pi \cdot \sqrt{\frac{a^3}{\mu_\odot}} \qquad \text{Perido Orbitale} \tag{1.2}$$

[3] *Astronomia Nova* (1609) e *Harmonices Mundi* (1619).

Politecnico di Torino

Questa relazione quantitativa tra il periodo e la dimensione dell'orbita è cruciale per la comprensione della struttura del sistema solare e ha applicazioni dirette nella determinazione delle masse dei corpi celesti attraverso l'analisi delle loro orbite.



Periastro :	$r_p = a(1-e)$	Apoastro :	$r_a = a(1+e)$
Semiasse Maggiore :	$a = \frac{r_p + r_a}{2}$	Semiasse Minore :	$b = \sqrt{a^2(1-e^2)}$
Eccentricità :	$e = \frac{r_a - r_p}{r_a + r_p}$	Semilatus Rectum :	$p = a(1-e^2)$

Figure 1.2: Rappresentazione delle leggi di Keplero

Le leggi di Keplero, frutto di un'attenta osservazione empirica dei movimenti dei pianeti nel sistema solare, forniscono una descrizione accurata delle loro traiettorie orbitali. Tali leggi, universalmente applicabili a qualsiasi entità sotto l'influenza di un corpo attrattore principale, tracciano le orbite in termini matematici, pur non delineando esplicitamente le forze fisiche sottostanti che governano tali movimenti.

È stato successivamente Isaac Newton a fornire la chiave interpretativa di questo meccanismo attraverso la sua formulazione della legge di gravitazione universale, elaborando così una spiegazione teorica che radica le osservazioni di Keplero in principi fisici universali.

Newton non solo ha matematizzato le leggi di Keplero, ma ha anche esteso il loro ambito di applicazione, dimostrando che la stessa forza che guida la caduta di una mela sulla Terra regola anche il moto dei pianeti intorno al Sole.

1.1.2 I Parametri Orbitali Classici

La descrizione delle orbite dei corpi celesti, come satelliti o pianeti, è effettuata attraverso l'uso di sei parametri orbitali classici. Questi parametri sono fondamentali per la determinazione precisa della posizione e della traiettoria di un oggetto nello spazio.

1. **Semiasse Maggiore** (a): Il semiasse maggiore è uno dei parametri più significativi in meccanica orbitale. Esso rappresenta la distanza media dal corpo centrale lungo l'asse maggiore dell'orbita ellittica e definisce la grandezza dell'orbita conica.
2. **Eccentricità** (e): L'eccentricità descrive la forma dell'orbita. Un valore di 0 corrisponde a un'orbita perfettamente circolare, mentre valori che si avvicinano a 1 indicano orbite estremamente ellittiche.
3. **Inclinazione** (i): L'inclinazione dell'orbita misura l'angolo tra il piano orbitale di un corpo e un piano di riferimento, come il piano equatoriale del corpo centrale.
4. **Longitudine del Nodo Ascendente** (Ω): La longitudine del nodo ascendente definisce l'angolazione orizzontale del nodo ascendente (NA) dell'orbita rispetto a un punto di riferimento fisso. Questo elemento stabilisce l'orientamento dell'orbita nel piano di riferimento.
5. **Argomento del Periastro** (ω): L'argomento del periastro è l'angolo, nel piano dell'orbita, tra il nodo ascendente e il punto di periastro. Questo parametro, combinato con la longitudine del nodo ascendente (Ω), descrive completamente l'orientamento dell'orbita nello spazio.
6. **Anomalia Vera all'Epoca** (ν): L'anomalia vera, simbolizzata con ν, è un parametro che indica la posizione di un corpo celeste lungo la sua orbita in un dato momento.

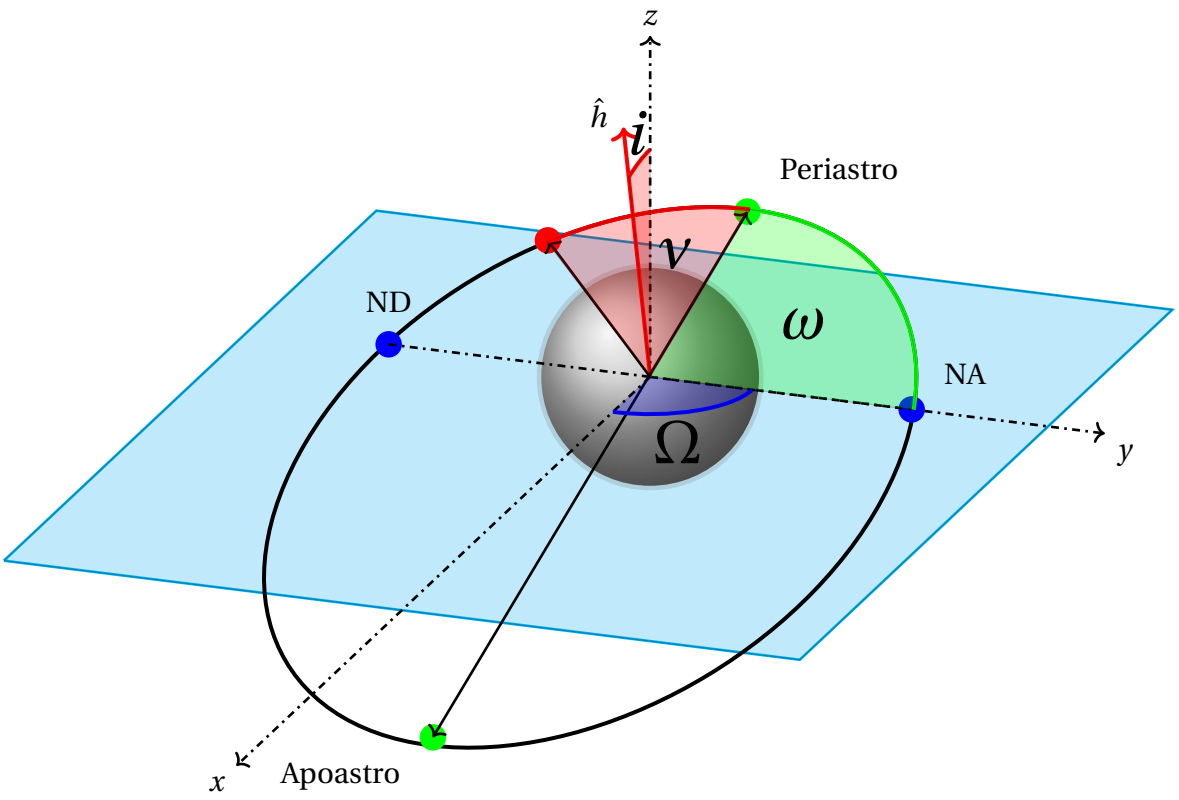


Figure 1.3: Rappresentazione dei Parametri Orbitali Classici.

Politecnico di Torino

1.2 IL PROBLEMA DEI DUE CORPI - *2-Body Problem*

La dinamica orbitale di uno *spacecraft* in orbita vicina alla Terra è governata principalmente dalla forza di gravità terrestre. Questa interazione, al cuore del problema dei due corpi, fornisce un quadro fondamentale per comprendere le traiettorie kepleriane degli oggetti in orbita. Tuttavia, per una descrizione più accurata e realistica del moto orbitale, è indispensabile considerare anche ulteriori effetti perturbativi, che, seppur di entità minore rispetto alla gravità terrestre, giocano un ruolo significativo nel determinare la precisione delle orbite e delle traiettorie spaziali.

Tra le forze perturbative che modificano i parametri orbitali di uno *spacecraft*, la non perfetta sfericità della Terra gioca un ruolo essenziale. La distribuzione di massa non uniforme del pianeta, risultante in una forma oblata, dà origine a variazioni nei campi gravitazionali vissuti dallo *spacecraft*, influenzando in modo particolare la longitudine[4] del nodo ascendente e l'argomento del perigeo[5]. Queste perturbazioni, caratterizzati dall'effetto J2, sono particolarmente significativi per *spacecraft* in orbite basse, dove la variazione del campo gravitazionale terrestre con l'altitudine è più pronunciata.

Le orbite a bassa e media quota (LEO e MEO) sono ulteriormente soggette a drag atmosferico, un effetto perturbativo che rallenta gradualmente lo *spacecraft*, riducendo la sua altitudine nel tempo. Questo effetto è dovuto alla presenza, seppur rarefatta, dell'atmosfera terrestre a queste altitudini, che agisce come una forza di resistenza al movimento dello *spacecraft*.

In aggiunta, la radiazione solare rappresenta un'altra fonte significativa di perturbazione, esercitando pressione e causando variazioni nella traiettoria. Questo effetto, sebbene generalmente meno intenso rispetto al drag atmosferico o all'influenza gravitazionale, può accumularsi nel tempo, portando a deviazioni significative dalle orbite previste, soprattutto per *spacecraft* con superfici ampie esposte alla radiazione solare o per missioni di lunga durata.

Con l'aumentare della distanza dalla Terra, le perturbazioni dovute alla gravità di altri corpi celesti diventano significative. La Luna, il Sole e, in misura minore, pianeti come Giove, esercitano influenze gravitazionali che possono alterare le traiettorie degli *spacecraft* in orbite più distanti o in missioni interplanetarie. Questi effetti, sebbene trascurabili per orbite strettamente vicine alla Terra, diventano vitali per la pianificazione di traiettorie e manovre in contesti più ampi del sistema solare.

La considerazione delle varie perturbazioni orbitali mette in rilievo il ruolo del problema dei due corpi come semplificazione ideale del contesto spaziale. Questo modello, nonostante sia una riduzione della complessità reale, si rivela estremamente utile per fornire una prima comprensione intuitiva e una soluzione analitica della dinamica orbitale. L'approccio basato sul problema dei due corpi consente di delineare le traiettorie orbitali con una precisione notevole in molti casi pratici, offrendo un punto di partenza solido per l'analisi più dettagliata necessaria in contesti specifici.

[4] Regressione per orbita diretta ($0 < i < \pi/2$) e progressione per orbita retrograda ($\pi/2 < i < \pi$).

[5] La linea degli apsidi non rimane fissa nel tempo e l'entità della sua variazione è funzione dell'inclinazione dell'orbita: precessione ($i < 63.4$ oppure $i > 116.6$), regressione ($63.4 < i < 116.6$) e nessuna perturbazione ($i = 63.4$ oppure $i = 116.6$)

Politecnico di Torino

1.2.1 EQUAZIONE E COSTANTI DEL MOTO

Sotto quindi le assunzioni di trascurare gli effetti perturbativi, considerare un unico corpo attrattore principale[6] e considerare i corpi come punti materiali, possiamo ricavare la soluzione dell'equazione del moto di uno spacecraft, espressa matematicamente come segue:

$$\ddot{\mathbf{r}} + \frac{\mu}{r^2}\hat{r} = 0 \qquad \text{Equazione del Moto} \tag{1.3}$$

Definiamo il parametro gravitazionale μ, come:

$$\mu = GM \quad \left[\frac{km^3}{s^2}\right] \qquad \begin{cases} \mu_\oplus = 3.98600 \cdot 10^5 \left[\frac{km^3}{s^2}\right] \\ \mu_\odot = 132712.44 \cdot 10^6 \left[\frac{km^3}{s^2}\right] \end{cases}$$

In virtù del fatto che il campo gravitazionale terrestre è conservativo, un oggetto in orbita, sottoposto esclusivamente all'influenza di tale campo, non subisce variazioni nella sua energia meccanica totale. Questo principio, radicato nelle leggi della fisica e matematicamente dimostrabile, assicura che lo *spacecraft* conservi un valore costante di energia meccanica lungo tutto il percorso della sua orbita. Tale energia è espressa dalla somma dell'energia cinetica, proporzionale al quadrato della velocità dell'oggetto, e dell'energia potenziale gravitazionale, inversamente proporzionale alla distanza dal centro di massa del corpo celeste attrattore.

$$\mathscr{E} = \frac{v^2}{2} - \frac{\mu}{r} = -\frac{\mu}{2a} \qquad \begin{matrix} \text{Energia Meccanica} \\ \text{Specifica} \end{matrix} \tag{1.4}$$

L'energia meccanica specifica dell'oggetto rimane costante in assenza di forze non conservative, quali la resistenza atmosferica o le spinte di propulsione, implicando che qualsiasi variazione nell'energia cinetica si riflette in una variazione equivalente ma opposta dell'energia potenziale. Di conseguenza, lo *spacecraft* alterna tra energia cinetica e potenziale durante il suo movimento orbitale, mantenendo però invariata la somma totale dell'energia meccanica.

La costanza dell'energia meccanica specifica riflette non solo un bilanciamento tra l'energia cinetica e potenziale dello *spacecraft*, ma è altresì un indicatore diretto della tipologia dell'orbita. Questo perché l'energia meccanica specifica è funzione del semiasse maggiore dell'orbita e del parametro gravitazionale del corpo attrattore principale. In altre parole, il valore dell'energia meccanica specifica non solo quantifica l'energia totale dello *spacecraft* in orbita, ma incorpora anche le informazioni fondamentali sulla geometria orbitale e sulla sua natura, sia essa circolare, ellittica, parabolica o iperbolica.

$$\begin{cases} \mathscr{E} < 0 & \text{Circonferenza o Ellisse} \\ \mathscr{E} = 0 & \text{Parabola} \\ \mathscr{E} > 0 & \text{Iperbole} \end{cases}$$

[6] Questo termine porta come ulteriore considerazione che la massa del corpo attrattore sia molto più grande della massa dell'oggetto orbitante.

Politecnico di Torino

Un'altra costante del moto orbitale, per il problema dei due corpi, è il momento della quantità di moto specifico. Questa costanza deriva dal fatto che, in un campo gravitazionale conservativo, l'accelerazione agente su uno *spacecraft* è sempre diretta radialmente verso il corpo centrale, conferendo alla traiettoria orbitale la proprietà di essere confinata in un piano. Questa relazione non solo sottolinea la conservazione del momento angolare specifico in assenza di forze non conservative, ma collega anche direttamente tale momento con le proprietà geometriche dell'orbita, come il suo semiasse maggiore e l'eccentricità.

$$|\vec{h}| = \sqrt{\mu \cdot p} = \sqrt{\mu \cdot a(1-e^2)} \qquad \begin{matrix}\text{Momento della Quantità} \\ \text{di Moto Specifica}\end{matrix} \tag{1.5}$$

Il piano orbitale, definito dal vettore posizione e dalla velocità dello *spacecraft*, rimane invariato nel tempo, assicurando che il moto sia planare. Il vettore del momento della quantità di moto specifico ($\hat{h}$) si posiziona perpendicolarmente a questo piano orbitale.

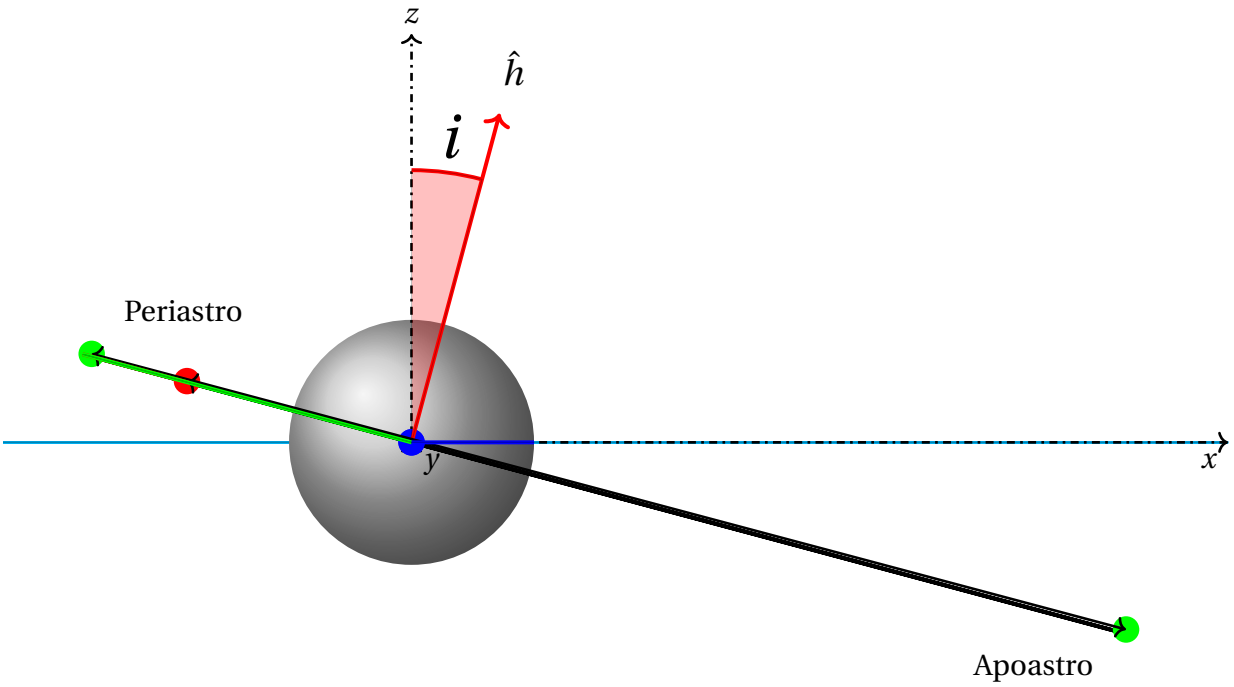


Figure 1.4: Rappresentazione del vettore Momento della Quantità di Moto.

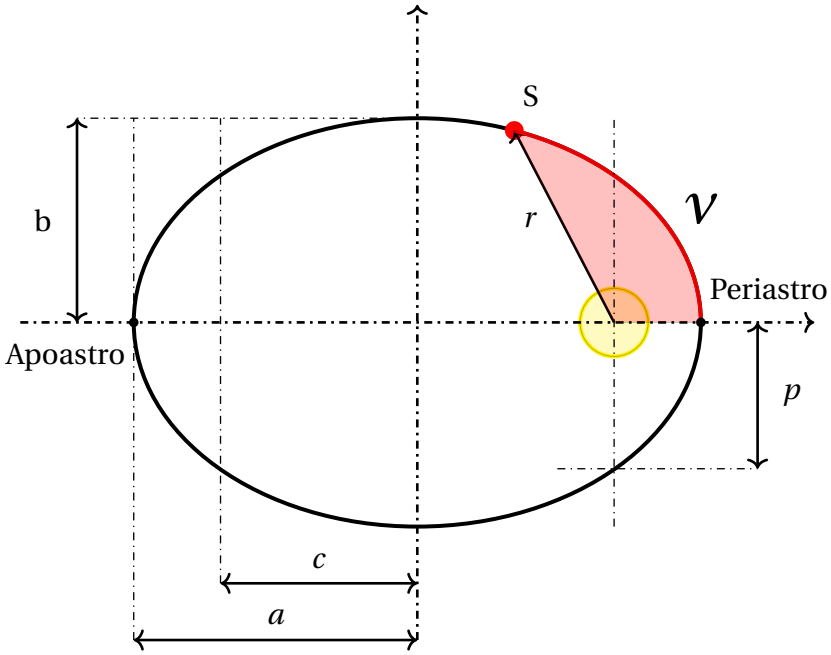
Politecnico di Torino

1.2.2 Equazione della Traiettoria

Dopo aver introdotto le quantità fondamentali, procediamo alla definizione matematica dell'equazione della traiettoria di un oggetto in orbita sotto l'influenza gravitazionale di un corpo centrale. Attraverso l'integrazione dell'equazione del moto nel contesto del problema dei due corpi, otteniamo l'espressione seguente per descrivere la distanza radiale in funzione dell'anomalia vera.

$$r = \frac{p}{1 + e \cdot cos\nu} = \frac{a(1 - e^2)}{1 + e \cdot cos\nu} \qquad \begin{matrix}\text{Equazione della} \\ \text{Traiettoria}\end{matrix} \tag{1.6}$$

La traiettoria orbitale si rivela così come una curva conica, la cui forma è determinata dal valore dell'eccentricità. Una conica è il luogo di punti in cui il rapporto fra la distanza dal fuoco, situato nel corpo centrale attrattore, e la distanza da una direttrice fissa, è costantemente uguale all'eccentricità dell'orbita.



Eccentricita	Tipo di Conica	$\mathscr{E}$	a
$e = 0$	Circonferenza	$\mathscr{E} < 0$	$a = 2r > 0$
$0 < e < 0$	Ellissi	$\mathscr{E} < 0$	$a > 0$
$e = 1$	Parabola	$\mathscr{E} = 0$	$a = \infty$
$e > 1$	Iperbole	$\mathscr{E} > 0$	$a < 0$

Figure 1.5: Equazione della Traiettoria.

Geometricamente, una conica (o sezione conica) corrisponde alla curva piana generata dall'intersezione di un cono circolare retto con un piano.

Politecnico di Torino

1.3 Traiettorie Interplanetarie

La navigazione interplanetaria trascende significativamente il modello kepleriano tradizionale, che postula l'esistenza di un solo corpo celeste dominante esercitante una forza gravitazionale. Nel contesto dei trasferimenti interplanetari, uno*spacecraft* si muove attraverso un ambiente in cui più corpi celesti, inclusi pianeti, lune e il Sole, esercitano influenze gravitazionali concorrenti, complicando notevolmente la dinamica del moto rispetto al problema a due corpi (*2-Body Problem*).

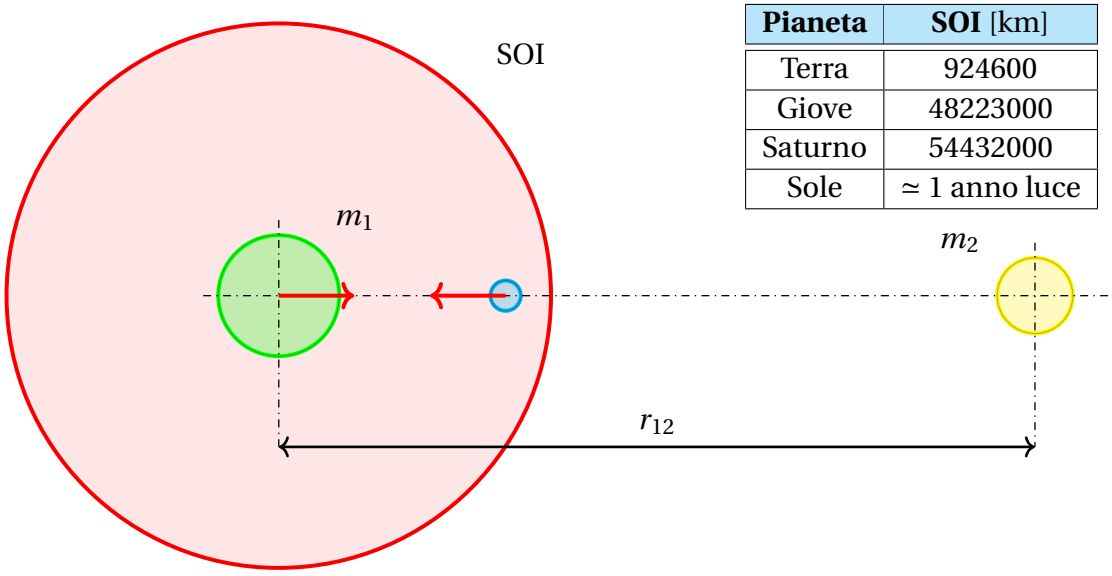
In questo intricato contesto dinamico, il moto dello *spacecraft* si configura come una traiettoria elaborata e caotica, in cui le forze gravitazionali dei vari corpi celesti si sovrappongono e interagiscono, alterando continuamente il percorso del veicolo spaziale. Questa complessità richiede un'analisi meticolosa e avanzata per garantire che la missione raggiunga il suo obiettivo con la massima precisione.

1.3.1 Sfera di Influenza

La trattazione delle trasferte interplanetarie può essere semplificata introducendo la definizione di *Sfera di Influenza* (*SOI*), ovvero la regione dello spazio entro cui la forza gravitazionale esercitata da un corpo celeste prevale su quella degli altri corpi. Questo concetto è fondamentale per determinare come e quando lo *spacecraft* risponde alle diverse forze gravitazionali in gioco.

$$SOI = r_{12} \cdot \left(\frac{m_2}{m_1}\right)^{2/5} \qquad \begin{matrix}\text{Sfera di}\\ \text{Influenza}\end{matrix} \tag{1.7}$$

Infatti, benché il moto di un corpo in un campo gravitazionale sia intrinsecamente non kepleriano a causa delle interazioni gravitazionali multiple, l'applicazione del modello kepleriano all'interno della SOI si rivela una convenzione analitica pragmatica. Inoltre la dimensione della SOI dipende da vari fattori, inclusi la massa del corpo celeste, la massa del corpo primario attorno a cui orbita, e la distanza relativa tra i due.



Pianeta	**SOI** [km]
Terra	924600
Giove	48223000
Saturno	54432000
Sole	≃ 1 anno luce

Figure 1.6: Rappresentazione della Sfera d'influenza (SOI).

La sfera di influenza è calcolata quindi sulla base del punto di equilibrio gravitazionale, dove le forze attrattive dei due corpi celesti si bilanciano.

Politecnico di Torino

1.3.2 PATCH CONICS APPROXIMATION

L'approssimazione tramite coniche raccordate, nota come metodo delle *patch conics*, rappresenta un'efficace strategia semplificata per l'analisi delle traiettorie interplanetarie. Questo approccio consente di decomporre il complesso problema del moto di uno *spacecraft* sotto l'influenza gravitazionale di più corpi celesti in una sequenza di problemi più gestibili, ognuno focalizzato sull'influenza predominante di un singolo corpo celeste alla volta.

Attraverso questa approssimazione, si assume che la traiettoria di uno *spacecraft* possa essere efficacemente rappresentata da segmenti di curve coniche (ellissi, parabole o iperboli) all'interno delle sfere di influenza (*SOI*) dei vari corpi celesti incontrati lungo il percorso. Questi segmenti di conica sono accuratamente raccordati ai confini delle *SOI*, garantendo una transizione fluida senza discontinuità lungo l'intero tragitto. La discontinuità è associata al cambio del parametro gravitazionale al varco delle *SOI*, che implica una variazione dell'energia meccanica specifica associata all'orbita, riflettendo il passaggio dall'influenza di un corpo celeste all'altro.

Il metodo delle *patch conics* si fonda sul presupposto che, in ogni tratto del percorso interplanetario, l'influenza gravitazionale dominante sia quella del corpo celeste più vicino. Questa semplificazione è ragionevole quando lo *spacecraft* si trova nettamente all'interno dell'*SOI* di un pianeta, dove le forze gravitazionali degli altri corpi celesti diventano trascurabili in prima approssimazione. Tuttavia, è importante riconoscere che questo metodo non tiene conto delle interazioni gravitazionali complesse e delle perturbazioni che possono sorgere quando più corpi esercitano una significativa influenza gravitazionale, come nelle vicinanze dei punti di Lagrange o durante i flyby multipli.

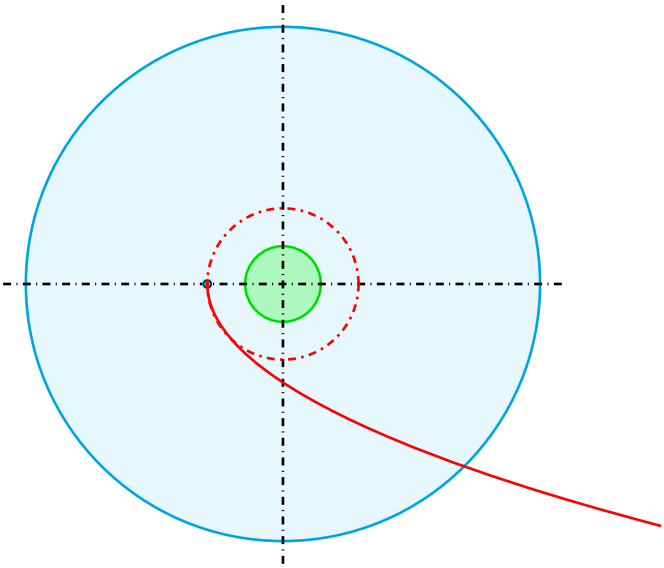


Figure 1.7: Fase di Fuga di una missione interplanetaria.

In sintesi, questo modo di procedere, può essere interpretato come l'attivazione o la disattivazione dell'influenza gravitazionale di specifici corpi celesti lungo il percorso dello *spacecraft*. Facilitando il calcolo delle traiettorie interplanetarie, il metodo delle *patch conics* si rivela particolarmente utile per la pianificazione preliminare delle missioni spaziali, offrendo una stima accurata del percorso con un notevole risparmio computazionale rispetto ai modelli più complessi che richiedono l'integrazione numerica delle equazioni del moto sotto l'influenza simultanea di più corpi celesti.

Politecnico di Torino

Una missione interplanetaria, analizzata attraverso questo metodo, viene quindi suddivisa in diverse fasi chiave:

1. **Fase di Fuga**: La fase in cui lo *spacecraft*, situato lungo una *parking orbit* attorno al corpo celeste di partenza, manovra fornendo il ΔV, necessario per fuggire dal corpo attrattore, seguendo una traiettoria che può essere approssimata da una parabola ($V_\infty = 0$) o un'iperbole ($V_\infty > 0$).
2. **Fase Eliocentrica**: La traiettoria del veicolo nello spazio interplanetario, principalmente sotto l'influenza del Sole, può essere rappresentata da un'ellisse o, in casi particolari, da una parabola o un'iperbole.
3. **Fase di Cattura**: Quando lo *spacecraft* entra nella sfera di influenza del corpo celeste target, la sua traiettoria viene nuovamente approssimata con una conica appropriata, che facilita il calcolo per l'inserimento in orbita o per il fly-by.

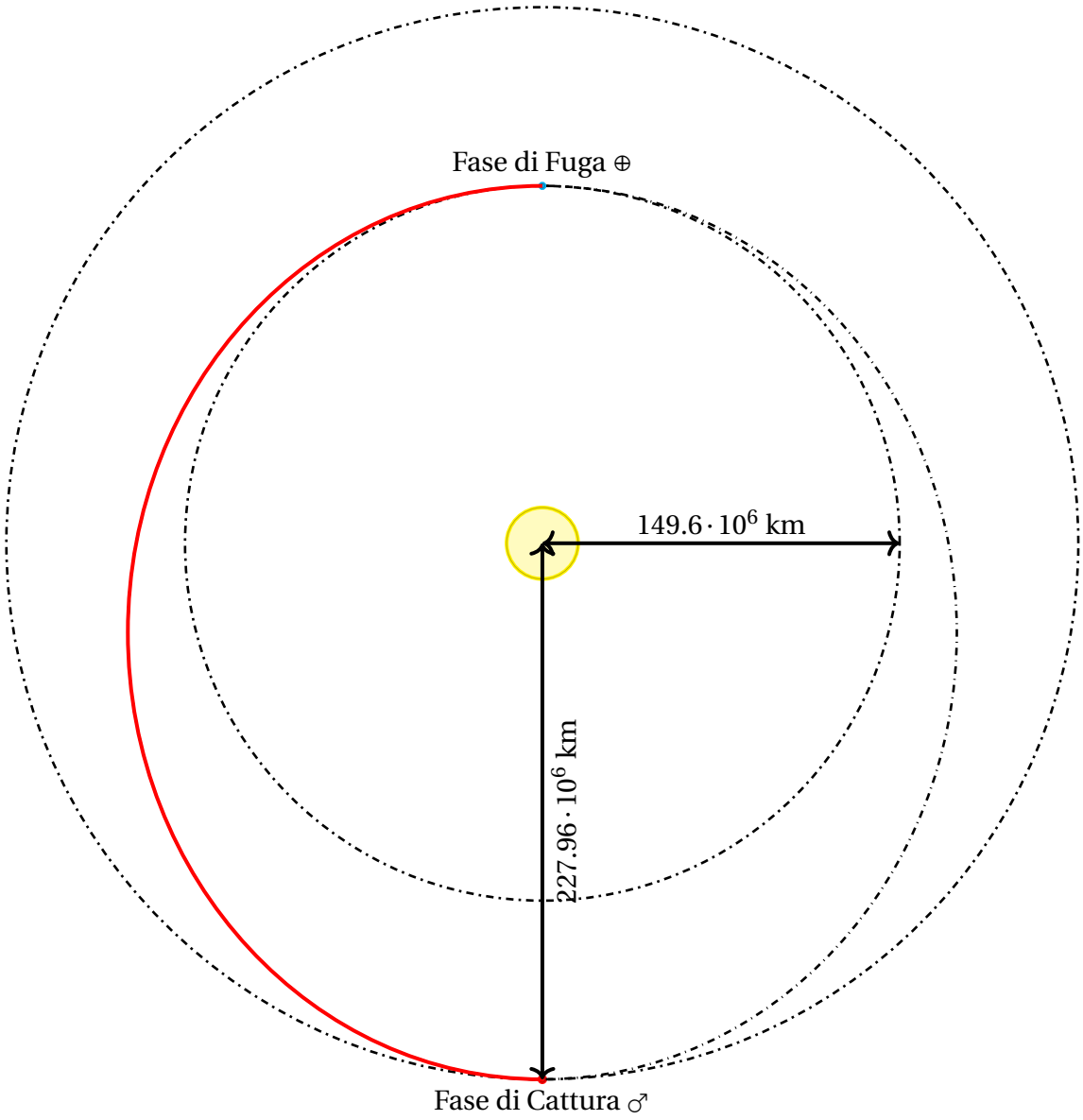


Figure 1.8: Rappresentazione del metodo *Patch Conics* per una trasferta interplanetaria Terra-Marte. La figura mostra la fase eliocentrica della missione, inoltre sono state rappresentate, in scala, le SOI dei vari pianeti.

Politecnico di Torino

La giustificazione dell'approssimazione impiegata nel metodo delle *patch conics* si basa su un'analisi attenta delle grandezze in gioco. È importante sottolineare che il valore della Sfera di Influenza (SOI) di un corpo celeste risulta essere solamente una frazione percentuale, generalmente inferiore al 7%, del semiasse maggiore dell'orbita del pianeta intorno al Sole.

Questa osservazione mette in luce come l'area di predominanza gravitazionale di un pianeta, benché critica per le manovre di cattura o di fuga, costituisca una porzione relativamente piccola dell'intero percorso di una missione interplanetaria.

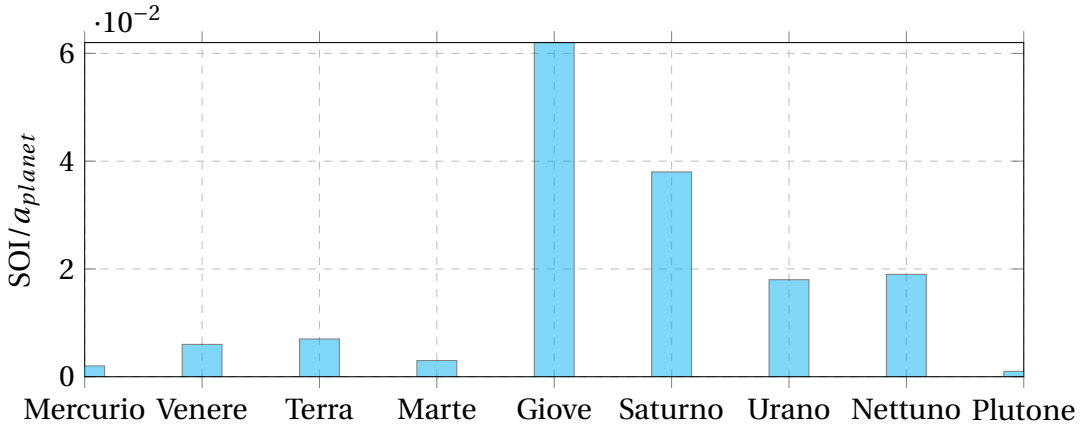


Table 1.1: Confronto fra le sfere d'influenza (SOI) dei vari pianeti del sistema solare e il semiasse maggiore della rispettiva orbita.

In contrasto, se si confronta il raggio della SOI con il raggio stesso del pianeta, la proporzione si inverte, evidenziando che la SOI estende ben oltre le dimensioni fisiche del corpo celeste. Questa differenza di scala sottolinea l'efficacia dell'approssimazione delle coniche raccordate: benché la SOI sia vasta rispetto al pianeta, essa rappresenta solo una minima parte dell'orbita totale di un veicolo spaziale che viaggia attraverso il sistema solare.

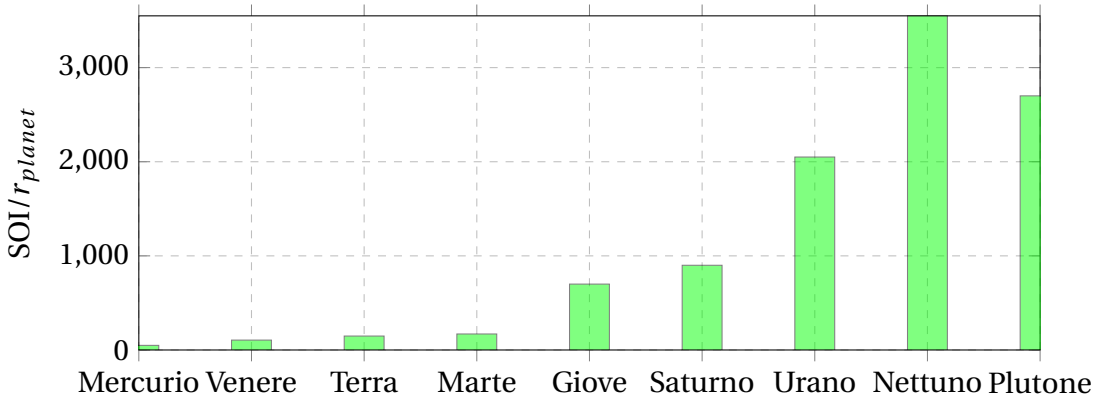


Table 1.2: Confronto fra le sfere d'influenza (SOI) dei vari pianeti del sistema solare ed il raggio medio del pianeta.

Il relativo ordine di grandezza della SOI in confronto al semiasse maggiore dell'orbita planetaria attorno al Sole giustifica l'applicazione dell'approssimazione delle coniche raccordate per semplificare l'analisi delle traiettorie interplanetarie.

Politecnico di Torino

MACHINE LEARNING

Politecnico di Torino

Politecnico di Torino

CHAPTER 2

INTRODUZIONE AL MACHINE LEARNING

2.1 PANORAMICA DEL MACHINE LEARNING

L'Intelligenza Artificiale (AI), nel suo senso più ampio, è definita dalla capacità di un sistema informatico di eseguire compiti che, tradizionalmente, presuppongono l'intelligenza umana. Tali compiti includono, ma non si limitano a, il riconoscimento di schemi, l'apprendimento basato su esperienze pregresse, l'adattamento a contesti mutevoli, l'elaborazione e risposta al linguaggio naturale, nonché la capacità di effettuare scelte complesse basate su un insieme articolato di criteri. L'ambizione dei ricercatori in questo ambito è di sviluppare sistemi capaci di operare queste funzioni in maniera autonoma ed efficiente, talvolta eccedendo le prestazioni umane in termini di rapidità ed accuratezza.

Il Machine Learning (ML), un pilastro fondamentale dell'Intelligenza Artificiale (AI), si distingue come il ramo di studio dedicato allo sviluppo di algoritmi capaci di apprendere autonomamente dai dati. Questa branca della AI si differenzia dall'approccio algoritmico tradizionale, caratterizzato da istruzioni esplicite per compiti specifici, in quanto gli algoritmi di ML evolvono attraverso l'analisi e l'interpretazione di grandi volumi di dati. Essi non solo identificano pattern e regolarità all'interno dei dati, ma li utilizzano per formulare previsioni o guidare il processo decisionale. Tale processo consente alle macchine di adattarsi autonomamente a nuove circostanze, aumentando in modo significativo la loro applicabilità in svariati campi.

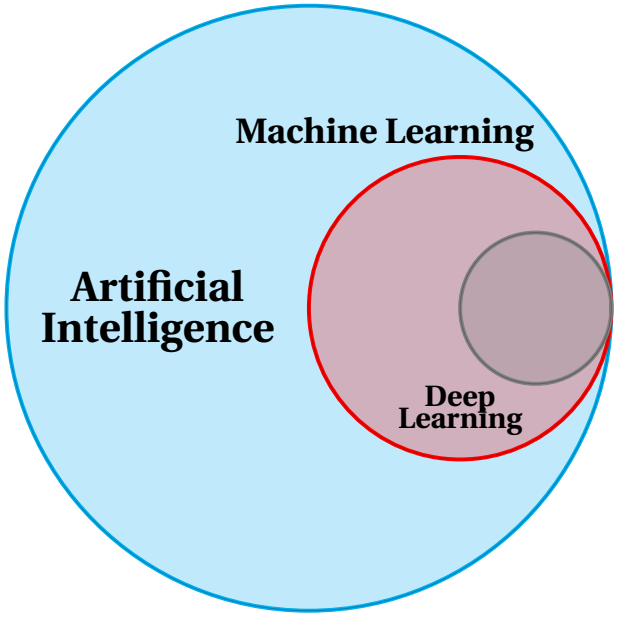


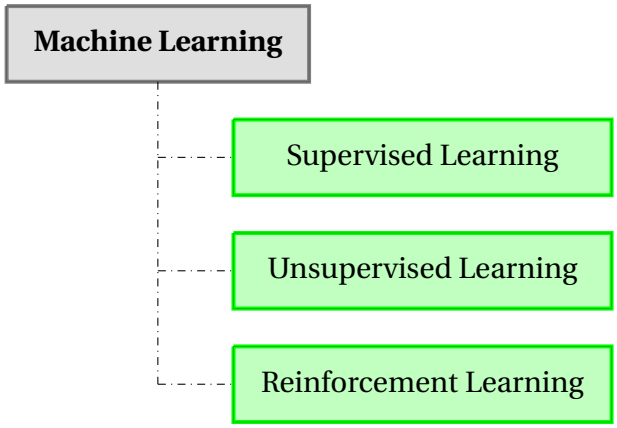
Figure 2.1: AI, Machine Learning and Deep Learning

Nel contesto del Machine Learning, l'apprendimento può essere descritto come il processo attraverso cui un algoritmo migliora la sua capacità di eseguire un compito specifico basandosi sui dati. Questi algoritmi, partendo da dati grezzi in input, esplorano lo spazio delle ipotesi (*hypothesis space*) per trovare una trasformazione funzionale che converta efficacemente questi dati in output significativi. Il fulcro di questo processo è la capacità di generalizzazione, ovvero l'abilità dell'algoritmo di applicare le conoscenze acquisite da un set di dati a situazioni nuove e non precedentemente incontrate.

Politecnico di Torino

Questa capacità di generalizzazione è fondamentale per la validità di un modello di ML. Un modello si comporta bene su dati noti ma fallisce nel generalizzare su dati nuovi è considerato sovradattato (*overfitted*). Al contrario, un modello che non riesce a catturare adeguatamente la complessità dei dati durante l'addestramento è sottoadattato (*underfitted*). Il successo di un algoritmo di ML è quindi misurato dalla sua capacità di bilanciare efficacemente queste due estremità, fornendo previsioni accurate e affidabili anche in condizioni non familiari.

In ML, la generalizzazione è l'obiettivo ultimo e più sfidante. Gli algoritmi sono progettati per apprendere dai dati di addestramento, ma il loro vero test è rappresentato dalla performance su dati non visti in precedenza. Questa capacità di adattamento non è solo una prova dell'efficacia dell'algoritmo, ma anche un indicatore della sua utilità pratica. In ultima analisi, un modello di ML efficace deve essere in grado di catturare l'essenza dei dati di addestramento e applicarla a nuovi dati, mantenendo un equilibrio tra adattabilità e affidabilità.



SUPERVISED LEARNING

Nell'ambito dell'Apprendimento Supervisionato (*Supervised Learning*), un algoritmo di Machine Learning viene addestrato utilizzando un dataset etichettato. Tale insieme di dati è composto da coppie di input e output corrispondenti, dove l'output funge da "etichetta". L'obiettivo primario è quello di istruire il modello al fine di predire l'output (o etichetta) per nuovi input, basandosi sulla correlazione appresa tra input e output all'interno del dataset di addestramento.

I problemi trattati nell'ambito del *Supervised Learning* possono essere categorizzati in due principali tipologie:

1. **Classificazione**: Questa categoria implica la previsione di etichette categoriche o discrete. Gli algoritmi di classificazione hanno lo scopo di assegnare ad ogni input una specifica categoria o classe, basandosi sulle caratteristiche distintive degli input.
2. **Regressione**: Si concentra sulla previsione di valori continui. In questi casi, i modelli sono volti a stimare una relazione funzionale tra variabili indipendenti (input) e dipendenti (output continui).

Tra le principali problematiche legate agli algoritmi di apprendimento supervisionato vi è il fenomeno dell'*overfitting*, in cui il modello apprende in modo eccessivamente preciso i dati di addestramento, perdendo di generalità.

Politecnico di Torino

Contrariamente agli approcci tradizionali, i modelli di Machine Learning (ML) supervisionati operano senza presupporre la conoscenza a priori delle leggi fisiche o logiche che governano il fenomeno oggetto di studio. Questi modelli vengono addestrati (*training*) su dataset ampi e vari, che includono sia dati di input che di output, consentendo al modello di scoprire e imparare le relazioni intrinseche tra questi.

Durante l'addestramento, algoritmi avanzati come la discesa del gradiente (*Gradient Descent*) vengono impiegati per affinare i parametri del modello. Ciò avviene attraverso la minimizzazione di funzioni di errore (*Loss Function*), che forniscono una misura dell'errore di predizione, permettendo al modello di estrapolare e generalizzare le relazioni osservate nei dati. Una volta che il modello è stato calibrato con precisione[1], esso diventa capace di applicare la conoscenza acquisita a situazioni reali e inedite, dimostrando una potente capacità di generalizzazione.

Un esempio storico che rispecchia l'essenza dell'apprendimento nel ML supervisionato è rappresentato dalle leggi di Keplero. Derivate da osservazioni empiriche dei movimenti dei corpi celesti, queste leggi furono successivamente raffinate da Keplero per meglio adattarsi ai dati astronomici che aveva raccolto. Questo processo, pur essendo empirico e basato sull'adattamento ai dati osservati, è analogo al modo in cui i modelli di ML supervisionato apprendono a descrivere e prevedere fenomeni complessi, basandosi esclusivamente sui dati a loro disposizione.

Per creare un modello di Machine Learning funzionale ed efficiente, si rendono indispensabili alcuni elementi fondamentali:

1. **Dataset:** Una collezione ben strutturata di input e output, categorizzata in *Training Set*, *Validation Set* e *Test Set*. Questo insieme di dati non solo fornisce la materia prima per l'addestramento, ma anche i mezzi per una valutazione accurata e obiettiva del modello.
2. **Metrica di Valutazione:** Un sistema di valutazione rigoroso per quantificare l'accuratezza del modello, confrontando gli output generati con quelli reali o attesi. Questa metrica è cruciale per misurare l'efficacia del modello e per guidarne l'ottimizzazione.
3. **Backpropagation e Ottimizzazione:** Processi fondamentali per l'aggiustamento e il miglioramento dei parametri del modello. La *backpropagation* permette al modello di apprendere dai propri errori, mentre l'ottimizzazione sistematica riduce l'errore complessivo indicato dalla *loss function*, rendendo il modello sempre più preciso e affidabile.

[1] Questo delicato equilibrio è influenzato da vari fattori, tra cui la qualità e la diversità del dataset, l'architettura del modello, e il numero adeguato di cicli di addestramento, evitando così il rischio dell'*overfitting*.

Politecnico di Torino

UNSUPERVISED LEARNING

L'Apprendimento Non Supervisionato (*Unsupervised Learning*) rappresenta una metodologia caratterizzata dall'addestramento di modelli su set di dati privi di etichette predefinite. A differenza dell'*Supervised Learning*, che si basa su dati con etichette note, l'*Unsupervised Learning* mira a scoprire strutture e schemi intrinseci nei dati, operando senza guide o etichette specifiche.

I principali ambiti di applicazione di questo tipo di apprendimento includono:

1. **Clustering**: Questo processo ha l'obiettivo di raggruppare insiemi di esempi basandosi sulla loro somiglianza intrinseca. Ad esempio, nel marketing, il clustering è utilizzato per identificare segmenti di clientela con comportamenti analoghi.
2. **Riduzione della Dimensionalità**: Consiste nel diminuire il numero di variabili casuali in esame. Ciò si basa sull'ipotesi che le informazioni rilevanti nei dati possano essere catturate tramite un insieme ridotto di nuove variabili.
3. **Apprendimento di Rappresentazioni**: Si focalizza sull'estrazione automatica delle caratteristiche essenziali dei dati, che possono rivelarsi utili in vari contesti, come la classificazione o la previsione nell'ambito dell'apprendimento supervisionato.

Nonostante le potenzialità, questa categoria di algoritmi presenta sfide e limitazioni. L'assenza di etichette predefinite può rendere l'interpretazione dei risultati soggettiva e complessa. Inoltre, la qualità dei dati di ingresso è di primaria importanza: dati rumorosi o incompleti possono portare a conclusioni errate. Inoltre, la scelta di parametri influisce significativamente sui risultati, richiedendo spesso un'accurata sperimentazione.

REINFORCEMENT LEARNING

L'Apprendimento per Rinforzo (*Reinforcement Learning*, RL) è un paradigma di apprendimento automatico in cui un agente impara a prendere decisioni ottimali interagendo con un ambiente. Il fulcro del RL è l'obiettivo dell'agente di massimizzare un cumulo di ricompense nel corso del tempo. Contrariamente all'apprendimento supervisionato, in cui il modello riceve esempi di comportamento ottimale, nel RL l'agente scopre autonomamente quali azioni generano le migliori ricompense attraverso la sperimentazione.

I componenti fondamentali dell'Apprendimento per Rinforzo includono:

1. **Agente**: L'entità che prende decisioni basate sulle informazioni ricevute.
2. **Ambiente**: Il contesto dinamico in cui l'agente opera e interagisce.
3. **Stati**: Rappresentazioni dell'ambiente che influenzano le decisioni dell'agente.
4. **Azioni**: Le decisioni o i movimenti che l'agente può intraprendere.
5. **Ricompense**: I feedback immediati ricevuti dall'agente dopo ogni azione, che guidano il suo apprendimento.

Una delle principali sfide nel reinforcement learning è il bilanciamento tra l'esplorazione di nuove azioni e l'ottimizzazione di quelle già note. Inoltre, alcuni algoritmi di RL possono presentare instabilità o difficoltà nella convergenza, e possono richiedere notevoli risorse computazionali, specialmente nel contesto del *Deep Reinforcement Learning*, ovvero quando gli algoritmi di RL sfruttano nella loro implementazione le informazioni ricavate da delle reti neurali, e prendono decisioni basandosi su di esse.

Politecnico di Torino

CHAPTER 3

DEEP LEARNING - DL

3.1 FONDAMENTI DEL DEEP LEARNING

Il Deep Learning (DL), una rivoluzionaria sotto-branca del Machine Learning (ML) che ha preso il sopravvento nel 2010, si contraddistingue per l'utilizzo di reti neurali profonde (*Deep Neural Networks* - DNN) e stratificate. Queste reti, costituite da molteplici strati di neuroni artificiali, conferiscono al DL la capacità di processare dati con un livello di profondità e complessità senza precedenti. A differenza del ML tradizionale, che tende a fare affidamento su modelli più lineari come le Support Vector Machines (SVM) o gli Alberi Decisionali (*Decision Trees*), il DL adotta un approccio radicalmente diverso.

Grazie alla sua architettura avanzata, il Deep Learning (DL) ha introdotto un cambiamento rivoluzionario nel campo dell'apprendimento automatico, specialmente nel processo di feature engineering. Quest'ultimo comprende la selezione, la manipolazione e la trasformazione delle variabili di input (o feature) per costruire modelli di machine learning, ha tradizionalmente richiesto un considerevole sforzo manuale e una conoscenza approfondita sia dei dati che del dominio applicativo. Questo processo implica lo sviluppo di feature che siano rappresentative, non ridondanti e significative per il problema specifico.

A differenza dei metodi tradizionali, in cui il feature engineering deve essere eseguito manualmente da esperti, le reti neurali nel deep learning possono autonomamente scoprire rappresentazioni di dati attraverso vari livelli di astrazione. Ciò significa che una rete neurale profonda è in grado di apprendere in modo indipendente le feature ottimali direttamente dai dati grezzi nel corso dell'addestramento.

Questa capacità diventa particolarmente evidente in architetture di deep learning come le Convolutional Neural Networks (CNNs) e le Recurrent Neural Networks (RNNs). Ad esempio, in una CNN utilizzata per l'elaborazione di immagini, gli strati iniziali possono imparare a identificare elementi semplici come bordi e texture, mentre gli strati più profondi possono riconoscere strutture più complesse come parti di oggetti o interi oggetti. In questo modo, la rete è capace di costruire una gerarchia di feature, evolvendo dalla semplicità alla complessità, senza la necessità di un intervento esterno per definire o selezionare manualmente queste caratteristiche.

I vantaggi di questo approccio sono molteplici. Innanzitutto, minimizza la quantità di lavoro manuale e la competenza necessaria per il feature engineering tradizionale. In secondo luogo, consente alle reti neurali di identificare e sfruttare relazioni complesse e non lineari nei dati che potrebbero sfuggire ai metodi tradizionali. Questo aspetto si rivela particolarmente vantaggioso in ambiti come la classificazione di immagini e l'elaborazione del linguaggio naturale, dove le feature rilevanti potrebbero non essere immediatamente evidenti o facilmente estraibili manualmente.

Nonostante l'innovativa autonomia del DL nella scoperta delle feature, la cura nella raccolta e nella pulizia dei dati rimane un aspetto fondamentale. La qualità, la varietà e la quantità dei dati raccolti giocano un ruolo essenziale nella formazione dei modelli di DL e nella loro capacità di generalizzare al di là dei dati di addestramento. Durante la raccolta, è quindi cruciale assicurarsi che i dati siano rappresentativi delle varie situazioni che il modello potrebbe incontrare.

Il processo di pulizia dei dati, che comprende l'eliminazione di duplicati, la correzione di errori, la gestione dei valori mancanti e l'identificazione degli outlier[1], è vitale per garantire l'affidabilità e la rappresentatività del dataset, minimizzando così il rischio di introduzione di bias o errori.

La profondità (*Depth*) di un modello di Deep Learning è determinata dal numero di strati (o *layers*) che lo compongono. Per questo motivo, in letteratura il DL è spesso descritto anche come *Layer Representation Learning* o *Hierarchical Representation Learning*, poiché ogni layer aggiuntivo aiuta a costruire una rappresentazione dei dati più stratificata e complessa. È importante notare che il numero di layer, e quindi il numero di parametri, è parte integrante dell'architettura del modello.

Tuttavia, è un errore comune presumere che un modello caratterizzato da un elevato numero di parametri sia necessariamente superiore rispetto a uno meno complesso. La decisione riguardante la profondità di un modello dovrebbe essere intrapresa considerando le peculiarità del problema specifico in esame. In linea di massima, la dimensione di un modello, ossia il numero dei suoi iperparametri, dovrebbe essere proporzionale alle dimensioni del dataset. Di conseguenza, un modello con un numero sostanziale di strati e neuroni è generalmente predisposto a esibire prestazioni migliori quando applicato a dataset di vasta entità.

Nonostante ciò, questa regola non riflette sempre fedelmente i risultati empiricamente osservabili. La ragione di questa discrepanza risiede nel fatto che molti aspetti restano ancora da chiarire e i modelli dimostrano una notevole sensibilità all'inserimento di ulteriori layer.

[1]Valori che si discostano in modo significativo dalla maggior parte degli altri dati nel dataset. Questi valori possono essere dovuti a errori di misurazione o di registrazione, a variazioni naturali estreme o a semplici anomalie.

Politecnico di Torino

3.1.1 TENSORI NEL DEEP LEARNING

I tensori sono una componente fondamentale del Deep Learning, rappresentando una unità fondamentale per la gestione dei dati. Essi rappresentano una generalizzazione delle matrici bidimensionali, estendendosi oltre le due dimensioni standard di righe e colonne per abbracciare strutture più complesse e multidimensionali. Questo permette ai tensori di catturare una vasta gamma di informazioni, rendendoli strumenti essenziali per il trattamento di dati complessi come immagini, suoni o testi.

I tensori sono solitamente composti da dati numerici[2], mentre in alcuni framework avanzati come TensorFlow, è anche possibile utilizzare stringhe di testo (*string*) come elementi dei tensori. La rappresentazione accurata dei dati di input sotto forma di tensori è cruciale nella creazione di modelli di Machine Learning efficaci.

La flessibilità dei tensori risiede nella loro struttura interna, che è caratterizzata da tre aspetti fondamentali: il *rank*, la *shape* e il *dtype*.

1. Il **Rank** di un tensore, talvolta riferito come ordine o grado, indica il numero di dimensioni presenti nel tensore. Ad esempio, uno scalare ha un rank di 0, un vettore ha un rank di 1, una matrice ha un rank di 2, e così via.

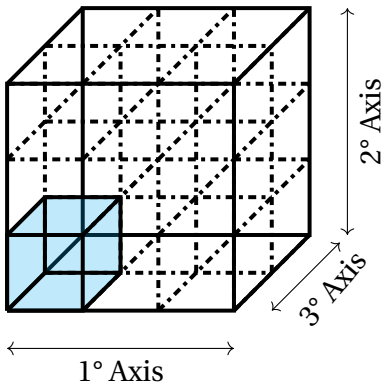
   In contesti di Deep Learning, è comune lavorare con tensori di rank superiore a 2, necessari per rappresentare dati complessi come immagini (altezza, larghezza, canali di colore) o video (numero di frame, altezza, larghezza, canali di colore).

2. La **Forma** di un tensore descrive la dimensione di ciascun asse del tensore. È una tupla di numeri interi che denota il numero di elementi lungo ogni dimensione. Per esempio, un tensore con una shape di (3, 4) rappresenta una matrice con 3 righe e 4 colonne. La shape è fondamentale per comprendere la forma del dato rappresentato dal tensore, come la risoluzione di un'immagine o la lunghezza di una sequenza temporale.

3. Il **Dtype**, o tipo di dati, di un tensore definisce il tipo di valori che il tensore può contenere. Questo può variare da numeri interi (*int16*, *int32*, *int64*) a numeri in virgola mobile (*float32*, *float64*) e altri tipi più specializzati. La scelta del dtype è influenzata dalla natura dei dati da rappresentare e dai requisiti computazionali, come la necessità di precisione nei calcoli o la limitazione delle risorse di memoria.

```
1 x = np.array([
2          [[1,2,3],   [11,22,33]],
3          [[4,5,36],  [44,55,66]],
4          [[7,8,9],   [77,88,99]]])
5 print(x.ndim, x.dtype, x.shape)
```

```
3 int32 (3, 2, 3)
```



Dal punto di vista informatico, ciò che distingue un tensore da un array numpy è la sua collocazione fisica. Infatti il tensore è ospitato direttamente sulla GPU.

[2]Tipicamente di tipo *float16*, *float32*, *float64*, o *unit8*.

Politecnico di Torino

3.2 Architettura di base del Deep Learning

Il Deep Learning, tramite le reti neurali, elabora dati complessi che ricordano il funzionamento del cervello umano. Queste reti, strutturate in diversi strati, o *layer*, processano l'informazione in modi unici e specializzati, ognuno contribuendo a un aspetto diverso dell'apprendimento. Questo processo può essere descritto come una sofisticata forma di distillazione, dove ogni *layer* della rete agisce selettivamente per filtrare e raffinare i dati input, consentendo alla rete di estrarre e accentuare caratteristiche essenziali e ridurre la complessità dell'informazione iniziale

I neuroni artificiali, al centro di queste architetture, sono più di semplici trasmettitori di dati. Come veri e propri interpreti, essi ricevono input, li analizzano attraverso pesi e bias e, mediante funzioni di attivazione, trasformano questi segnali in output significativi. Questo processo non è statico ma si evolve con l'apprendimento, adattandosi e migliorando continuamente.

Le reti neurali si manifestano in diverse configurazioni, a seconda delle esigenze specifiche. I layer densamente connessi (*Dense* o *Fully Connected*), per esempio, formano una rete complessa in cui ogni neurone interagisce con tutti quelli dello strato precedente, creando una mappa dettagliata delle relazioni tra i dati. I layer convolutivi, d'altra parte, si focalizzano su aree locali degli input, come nel riconoscimento di pattern visivi, dove l'analisi di texture e contorni è fondamentale.

I layer ricorrenti, o *RNN*, sono progettati per gestire dati sequenziali, come il linguaggio parlato o testuale. In queste strutture, l'informazione viaggia non solo in avanti ma anche all'indietro, permettendo alla rete di integrare il contesto passato e presente. Questa caratteristica rende le RNN strumenti potenti per applicazioni come il riconoscimento vocale e la generazione di testo.

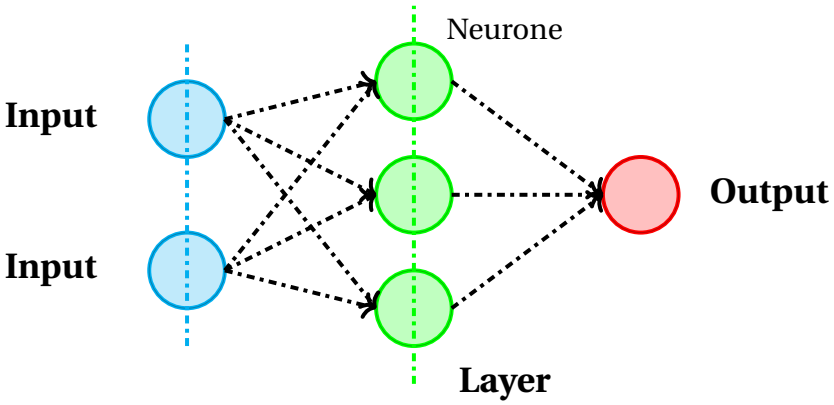


Figure 3.1: Esempio di una Neural Networks - *Fully Connected*

1. **Neuroni**: Elementi fondamentali di un *layer*. Ogni neurone riceve input, esegue una somma pesata su questi input, applica una funzione di attivazione, e produce un output.
2. **Weight e Bias**: Determinanti nell'influenzare il comportamento del neurone nel calcolo dell'output.
3. **Funzione di Attivazione**: Una funzione matematica che introduce non-linearità nel processo, permettendo di catturare relazioni complesse nei dati.

Politecnico di Torino

3.2.1 COME FUNZIONA UN MODELLO DI DEEP LEARNING

Un modello di deep learning trasforma i dati di input in output attraverso vari livelli (o *layers*), in modo automatico. Il modello cerca di trovare la trasformazione ottimale che elabora i dati di input in modo da restituire una previsione (*Prediction*) il più vicina possibile alla realtà. Questo processo di ottimizzazione è reso possibile da un segnale di feedback che guida l'*optimizer* nella modifica dei parametri del neurone, noto come algoritmo di *backpropagation*.

Come abbiamo già accennato, le *deep neural networks* elaborano i dati di input per produrre i dati di output mediante una serie di trasformazioni semplici che avvengono all'interno dei vari layer. Queste trasformazioni sono il risultato dell'apprendimento del modello durante la fase di *Training* e validazione. Ciò implica che le caratteristiche distintive di un layer, ovvero il suo ruolo nella trasformazione del dato di input, sono codificate nei parametri (θ: bias, ϕ: pesi) del layer stesso.

Più specificamente, queste caratteristiche si riflettono nei *weight* (pesi) e nei *bias* (valori di bias)[3] dei neuroni all'interno del layer. I pesi regolano l'importanza delle input features nella determinazione dell'output, mentre i bias permettono di aggiustare l'output anche in assenza di input significativi.

Essenzialmente, l'apprendimento in una rete neurale profonda si concentra sul trovare i valori ottimali dei pesi e dei bias per ogni neurone all'interno dei vari layer. Questi valori permettono al modello di generalizzare[4] al di là dei dati specifici su cui è stato addestrato, affrontando efficacemente problemi all'interno dello spazio dei dati per cui è stato allenato. Questo processo mira a rendere il modello versatile e adattabile, pur tenendo conto dei limiti imposti dalla natura e dalla varietà dei dati di training.

Entrando più nel dettaglio andiamo a capire quali siano i principali agenti protagonisti all'interno di una deep neural network. Possiamo infatti andare a definire:

1. **Dataset**: Il nucleo informativo su cui si basa la fase di addestramento e validazione.
2. **Layer e Neuroni**: Le unità di elaborazione che eseguono i calcoli e le trasformazioni dei dati.
3. **Loss Function**: Il sensore che misura l'errore e che orienta l'ottimizzazione del modello.
4. **Optimizer**: Il responsabile dell'aggiustamento dei pesi e dei bias.

Oltre a questi elementi, vi è un continuo scambio di informazioni e regolazioni all'interno della rete, un dialogo incessante tra input e output, errori e aggiustamenti. Questa moltitudine di calcoli e correzioni è ciò che permette ai modelli di Deep Learning di adattarsi, apprendere e, infine, eccellere nelle loro prestazioni.

[3] Il numero di parametri in un modello di DL si riferisce al numero totale di pesi e bias che il modello deve imparare durante l'addestramento.

[4] Ciò si riferisce alla capacità del modello di eseguire predizioni accurate su dati di input che differiscono da quelli utilizzati nelle fasi di training, validation e test.

Politecnico di Torino

Il ciclo iterativo, durante l'allenamento di un modello di DL, può essere schematizzato come segue. Inizialmente, il dato in input è processato sequenzialmente attraverso i vari *layer* del modello. Questa progressione attraverso le stratificazioni neurali culmina nella generazione di una *predizione* (Y'). La fase successiva implica un confronto critico tra il valore predetto (Y') e il valore effettivo, denominato *true target*, che rappresenta la realtà di riferimento all'interno del dataset. Questo confronto è valutato da una *metrica*, definita in base all'architettura dl modello.

Il risultato di tale confronto è sintetizzato nel *loss score*, un indicatore quantitativo della discrepanza tra predizione e realtà. Questo punteggio di perdita è intrinsecamente legato al valore degli *iperparametri* del modello[5]. In risposta a ciò, attraverso un meccanismo noto come *optimizer*, la rete neurale procede all'autoregolazione dei propri iperparametri. Quest'ultima azione ha lo scopo primario di minimizzare il *loss score*, ottimizzando così le prestazioni del modello in un processo di apprendimento continuo e dinamico.

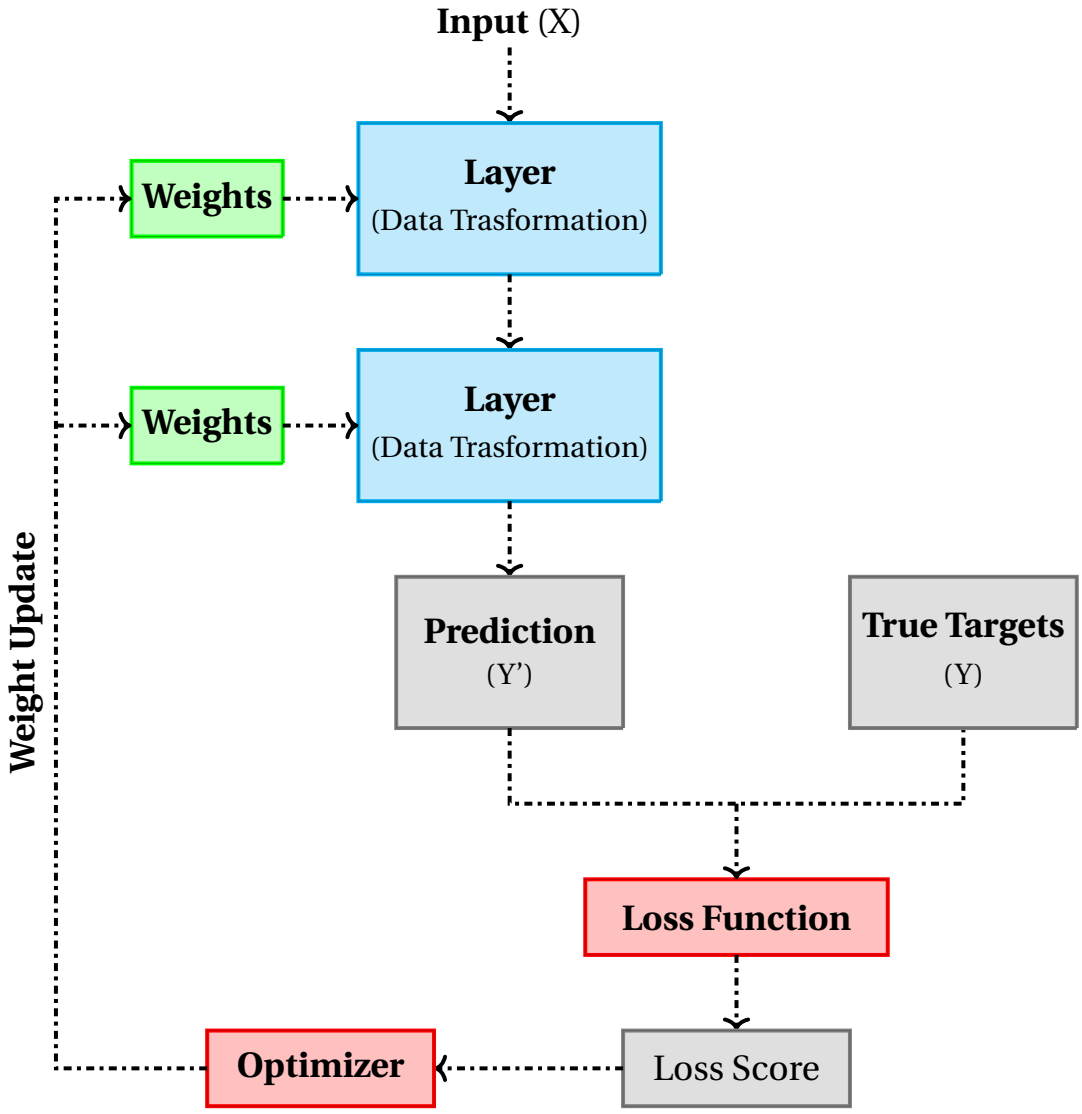


Figure 3.2: Deep Learning Neural Network - 2 Layers

[5]La somma complessiva di tutti i pesi e bias, funzione diretta del numero di layer e di neuroni presenti nella rete.

3.2.2 BackPropagation Algorithm

La *backpropagation*, o retropropagazione dell'errore è l'algoritmo che permette alle reti di apprendere efficacemente dai loro errori, aggiustando in maniera mirata i pesi e i bias per affinare le prestazioni.

La rete neurale può essere considerata come un insieme di strati neuronali interconnessi, dove ogni neurone processa gli input e invia l'output al successivo strato. L'addestramento si compone, quindi, di due fasi cruciali: la propagazione in avanti (*forward propagation*) e la retropropagazione (*backpropagation*).

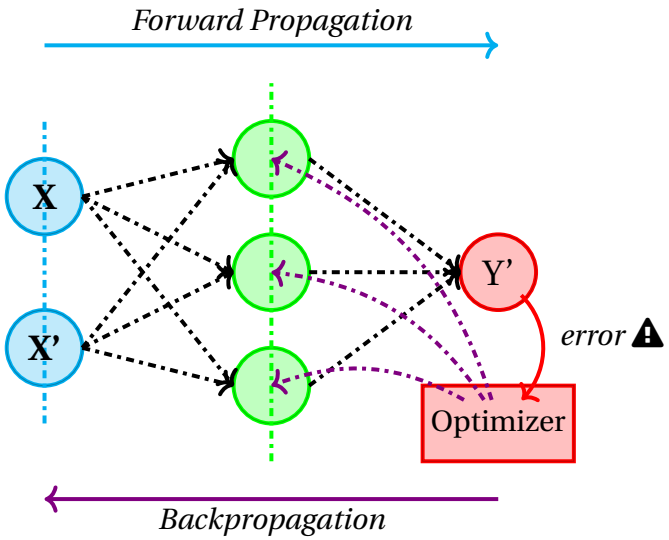


Figure 3.3: Forward e Backward propagation.

Nella fase di forward propagation, l'input attraversa la rete, venendo elaborato strato dopo strato, fino a generare un output finale. Quest'ultimo viene confrontato con l'output atteso, e la discrepanza tra i due è valutata come errore.

Dopo questa fase, entra in scena la backpropagation. L'obiettivo è ridurre l'errore complessivo della rete, regolando i pesi e i bias dei neuroni. Questo processo si articola in diverse tappe:

1. **Calcolo dell'Errore**: Si misura l'errore utilizzando una *loss function*, come la *mean squared error* per problemi di regressione o la *cross-entropy loss* per problemi di classificazione.
2. **Backpropagation**: L'errore viene retropropagato dalla fine all'inizio della rete. In questa fase, si calcola il gradiente della *loss function* rispetto a ogni peso e bias, determinando il loro contributo all'errore totale.
3. **Aggiornamento dei Pesi e dei Bias**: Tramite tecniche come la *gradient descent*, i pesi e i bias vengono aggiornati in modo da minimizzare l'errore. Il tasso di apprendimento (*Learning Rate*) regola l'entità di questi aggiustamenti.
4. **Iterazione del Processo**: Il procedimento viene ripetuto diverse volte durante le epoche, utilizzando l'intero dataset di addestramento, fino al raggiungimento di un miglioramento trascurabile o di un criterio di arresto prefissato.

Politecnico di Torino

3.3 NEURAL NETWORK - NN

Le reti neurali superficiali, note anche come *Shallow Neural Networks*, si distinguono per la loro architettura caratterizzata da un singolo strato nascosto (*hidden layer*). Questa struttura semplice le rende efficaci nell'approssimare funzioni non lineari, con una complessità minore rispetto alle reti neurali profonde, o *Deep Neural Networks*.

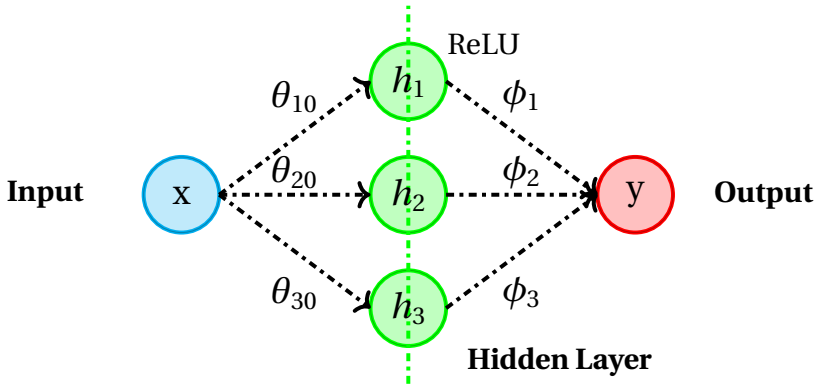


Figure 3.4: Shallow NN

Una shallow neural network è essenzialmente una funzione $y = f[x,\phi]$, che trasforma un input scalare (x) in un output scalare (y), con un grado di profondità n generico. Il grado di profondità di questa rete dipende dal numero di *hidden units*, ovvero i neuroni appartenenti al singolo hidden layer, il cui numero caratterizza la *networks capacity*.

$$\begin{aligned} y = f[x,\phi] &= \phi_0 + \phi_1 \cdot a[\theta_{10} + \theta_{11} \cdot x] + \ .... \ + \phi_n \cdot a[\theta_{n0} + \theta_{n1} \cdot x] \\ &= \phi_0 + \sum_{i=1}^{n} \phi_n \cdot h_n \qquad h_n = a[\theta_{n0} + \theta_{n1} \cdot x] \qquad \text{Shallow Neural Networks} \end{aligned} \tag{3.1}$$

In questa formulazione, ϕ rappresenta i pesi della rete, θ i bias, e $a[\cdot]$ indica la funzione di attivazione. Il processo di calcolo in una rete neurale superficiale si scompone in tre fasi distinte:

1. Inizialmente, si calcolano le equazioni lineari dei dati di input ($h_i = \theta_{i0} + \theta_{i1} \cdot x$), definite come *hidden units*.
2. Successivamente, questi valori vengono elaborati dalla funzione di attivazione ($a[\cdot]$).
3. Infine, i risultati vengono combinati con i pesi corrispondenti (ϕ_i) aggiungendo un termine costante, l'offset (ϕ_0).

$$\begin{cases} h_1 = \theta_{10} + \theta_{11} \cdot x \\ h_2 = \theta_{20} + \theta_{21} \cdot x \\ \vdots \\ h_n = \theta_{n0} + \theta_{n1} \cdot x \end{cases} \longrightarrow a \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix} \longrightarrow y = \phi_0 + \phi_1 \cdot a[h_1] + \ ... \ \phi_n \cdot a[h_n]$$

Politecnico di Torino

Considerando una shallow NN caratterizzata da 3 *hidden units*, a parametri ϕ e θ fissati, possiamo andare a valutare come si comporta il workflow della rete per andare a creare in output la funzione approssimata.

```
import matplotlib.pyplot as plt
import numpy as np

# Define the input range
x = np.linspace(-1, 2, 100)

# Define the values for theta and phi
theta = np.array([[-0.5, 0.5], [0.8, -1.2], [-0.1, 0.4]])
phi = np.array([-0.6, -0.4, 0.6, 0.2])

# Calculate the linear functions
h = np.array([theta[i][0] + theta[i][1] * x for i in range(3)])

# Apply ReLU activation function
a = np.maximum(0, h)

# Combine the hidden units with the weights and add an offset
offset = phi[0]
f = offset + np.dot(phi[1:], a)
```

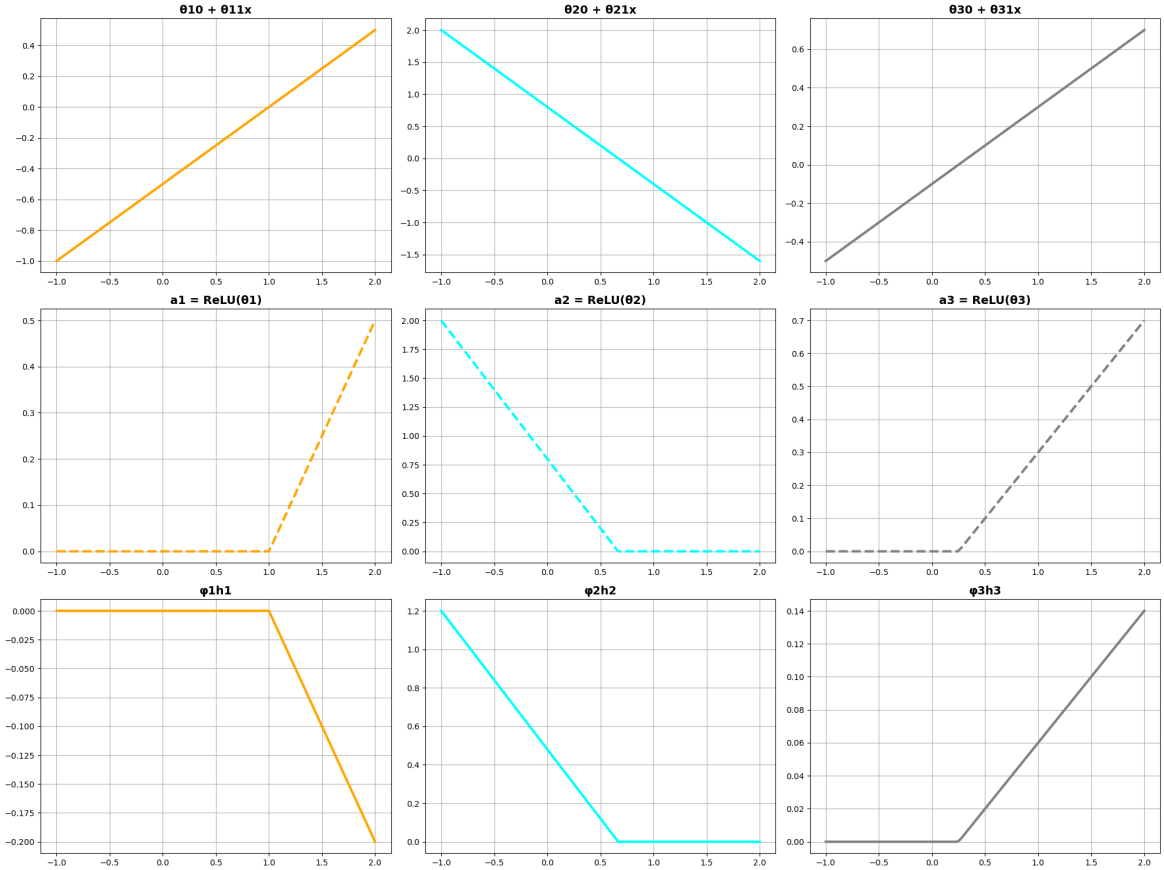


Figure 3.5: Processo di funzionamento di una Shallow Neural Networks.

Politecnico di Torino

Le reti neurali superficiali[6] sono capaci di rappresentare una vasta famiglia di funzioni, determinata dai parametri (ϕ,θ), consentendo di approssimare diversi comportamenti dei dati. Infatti avendo a disposizione abbastanza capacità nella rete (*network capacity*), ovvero un numero adeguato di hidden units, le shallow networks possono descrivere qualsiasi funzione continua 1D.

Considerando infatti una rete superficiale generica, caratterizzata da un numero n di *hidden units*.

$$y = \phi_0 + \sum_{i=1}^{n} \phi_n \cdot h_n \qquad h_n = a[\theta_{n0} + \theta_{n1} \cdot x]$$

Questa rete riuscirà ad approssimare una funzione, costruendo una funzione lineare a tratti avente n punti di giunzione ed $n+1$ tratti di funzioni lineari. Quindi, ogni volta che sarà aggiunto una hidden unit alla rete, alla funzione in output sarà aggiunto in tratto lineare.

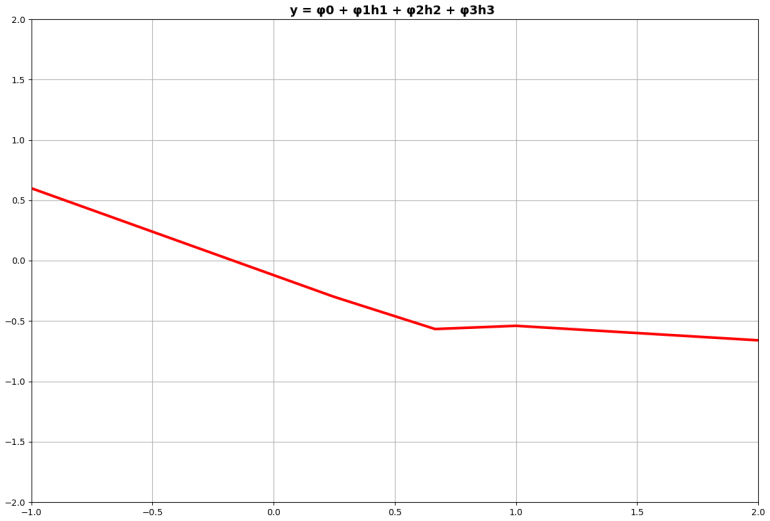


Figure 3.6: Funzione in output di una Shallow Neural Networks.

Grazie all'*universal approximation thorem* possiamo quindi affermare che per ogni funzione continua deve esistere una shallow neural networks che può approssimarla con una specifica precisione.

Conoscendo i parametri della rete, è possibile effettuare predizioni o inferenze per un dato input. Questo processo rappresenta il nucleo del funzionamento di una rete neurale, dove l'apprendimento e l'aggiustamento dei parametri avviene durante la fase di allenamento.

[6] Le reti neurali con almeno un layer sono anche dette *multi-layer perceptron* o MLP.

Politecnico di Torino

L'evoluzione logica delle shallow neural networks consiste nell'aggiungere un numero arbitrario di layers alla rete, andando quindi creare le *Deep Neural Networks*. Infatti, sebbene aumentando il numero di neuroni di una shallow neural networks aumentiamo anche il grado di approssimazione della stessa, le reti neurali profonde permettono di avere delle performance superiori, a numero di parameri fissati.

Le reti neurali profonde (*Deep Neural Networks*, DNN) rappresentano un'evoluzione significativa delle architetture neurali. La loro caratteristica distintiva è l'elevato numero di strati nascosti (*hidden layers*), che consente loro di apprendere rappresentazioni di dati a vari livelli di astrazione.

Essenzialmente possiamo vedere una Deep Neural Networks, caratterizzata da due hidden layer, come l'unione in serie di due Shallow Neural Networks, in cui:

$$y = \phi_0 + \phi_1 \cdot h_1 + \phi_2 \cdot h_2 + \phi_3 \cdot h_3 \qquad y' = \phi'_0 + \phi'_1 \cdot h'_1 + \phi'_2 \cdot h'_2 + \phi'_3 \cdot h'_3$$

$$\begin{cases} h_1 = a[\theta_{10} + \theta_{11} \cdot x] \\ h_2 = a[\theta_{20} + \theta_{21} \cdot x] \\ h_3 = a[\theta_{30} + \theta_{31} \cdot x] \end{cases} \qquad \begin{cases} h'_1 = a[\theta'_{10} + \theta'_{11} \cdot y] \\ h'_2 = a[\theta'_{20} + \theta'_{21} \cdot y] \\ h'_3 = a[\theta'_{30} + \theta'_{31} \cdot y] \end{cases}$$

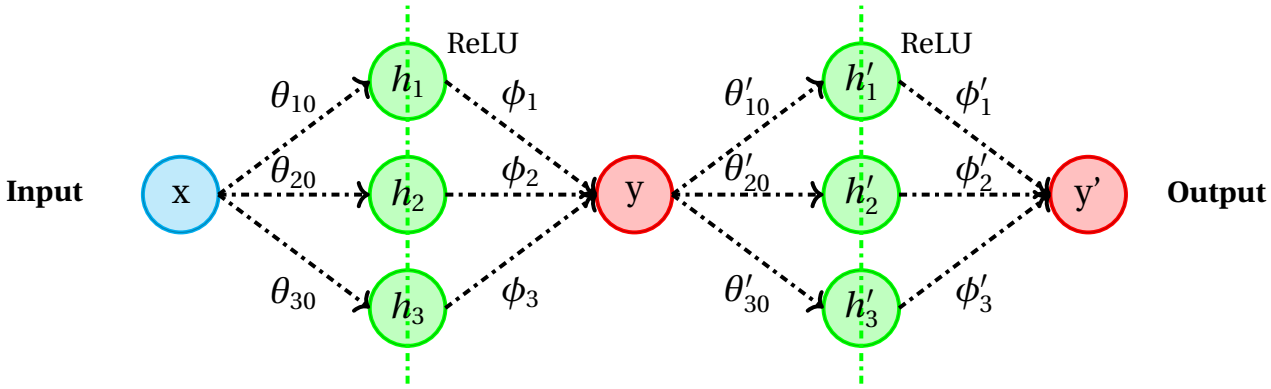


Figure 3.7: Unione in serie di due Shallow NN.

Di conseguenza:

$$\begin{cases} h'_1 = a[\theta'_{10} + \theta'_{11} \cdot y] = a[\theta'_{10} + \theta'_{11}\phi_0 + \theta'_{11}\phi_1 h_1 + \theta'_{11}\phi_2 h_2 + \theta'_{11}\phi_3 h_3] \\ h'_2 = a[\theta'_{20} + \theta'_{21} \cdot y] = a[\theta'_{20} + \theta'_{21}\phi_0 + \theta'_{21}\phi_1 h_1 + \theta'_{21}\phi_2 h_2 + \theta'_{21}\phi_3 h_3] \\ h'_3 = a[\theta'_{30} + \theta'_{31} \cdot y] == a[\theta'_{30} + \theta'_{31}\phi_0 + \theta'_{31}\phi_1 h_1 + \theta'_{31}\phi_2 h_2 + \theta'_{31}\phi_3 h_3] \end{cases}$$

Riscriviamo in questo modo:

$$\begin{cases} h'_1 = a[\psi_{10} + \psi_{11} h_1 + \psi_{12} h_2 + \psi_{13} h_3] \\ h'_2 = a[\psi_{20} + \psi_{21} h_1 + \psi_{22} h_2 + \psi_{23} h_3] \\ h'_3 = a[\psi_{30} + \psi_{31} h_1 + \psi_{32} h_2 + \psi_{33} h_3] \end{cases}$$

Unendo in serie due Shallow neural networks caratterizzate da tre neuroni ciascuna si nota come il numero di regioni lineari risultanti dalla funzione in output sia di gran lunga superiore all'output di una shallow neural networks avente sei neuroni.

Politecnico di Torino

Possiamo quindi vedere l'unione in serie di due reti neurali superficiali come una *deep neural network* avente due *hidden layer*.

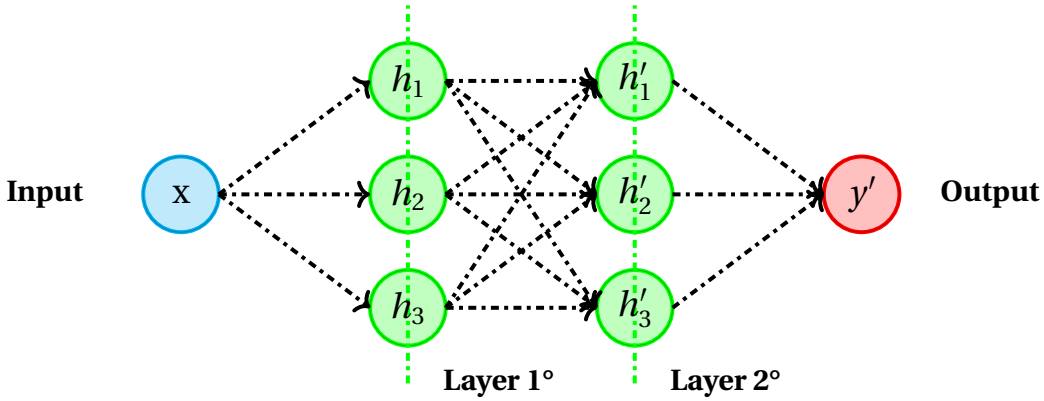


Figure 3.8: Deep NN

La funzione in output è la medesima, e dal momento che le operazioni all'interno di queste reti consistono in trasformazioni lineari alternate con le funzioni di attivazione, possiamo riscrivere il tutto utilizzando una notazione matriciale nel seguente modo:

$$\begin{Bmatrix} h_1 \\ h_2 \\ h_3 \end{Bmatrix} = \mathbf{a}\left[\begin{Bmatrix} \theta_{10} \\ \theta_{20} \\ \theta_{30} \end{Bmatrix} + \begin{Bmatrix} \theta_{11} \\ \theta_{21} \\ \theta_{31} \end{Bmatrix} \cdot x\right]$$

$$\begin{Bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{Bmatrix} = \mathbf{a}\left[\begin{Bmatrix} \psi_{10} \\ \psi_{20} \\ \psi_{30} \end{Bmatrix} + \begin{bmatrix} \psi_{11} & \psi_{12} & \psi_{13} \\ \psi_{21} & \psi_{22} & \psi_{23} \\ \psi_{31} & \psi_{32} & \psi_{33} \end{bmatrix} \cdot \begin{Bmatrix} h_1 \\ h_2 \\ h_3 \end{Bmatrix}\right]$$

Di conseguenza la DNN rappresenta la funzione seguente:

$$y' = \phi'_0 + \{\phi'_1 \quad \phi'_2 \quad \phi'_3\} \cdot \begin{Bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{Bmatrix} \tag{3.2}$$

Passando quindi da una rete neurale con un solo hidden layer a una con più layers, incrementiamo la complessità e la capacità della rete di modellare relazioni complesse e astratte. Infatti la *capacity* di una rete neurale è funzione del numero di hidden units[7] (neuroni) che possiede, e di conseguenza anche del numero di hidden layers.

L'aggiunta di *layers* aggiuntivi consente alla rete di estrarre caratteristiche a livelli diversi di astrazione, migliorando la sua capacità di interpretare e classificare i dati in modo più efficace. Questa caratteristica risulta evidente andando a considerare, in funzione degli iperparametri, le regioni lineari approssimate delle due tipologie di reti.

Shallow Neural Networks - SNN

$$\begin{cases} D > 2 & \text{Neuroni} \\ K = 1 & \text{Layer} \\ 3D+1 & \text{Parametri} \end{cases}$$

$D+1$ Linear Regions

Deep Neural Networks - DNN

$$\begin{cases} D > 2 & \text{Neuroni} \\ K & \text{Layer} \\ 3D+1+(K-1)D(D+1) & \text{Parametri} \end{cases}$$

$(D+1)^K$ Linear Regions

[7] In letteratura il numero di *layer* ed il rispettivo numero di neuroni associati a quel layer sono noti col nome di *hyperparameters*.

Politecnico di Torino

3.3.1 Reti Neurali Feed-forward

Le reti neurali *feed-forward* si distinguono come la forma più basilare, eppure incredibilmente performante, di reti neurali. Queste reti sono caratterizzate da un flusso di informazioni che procede in una direzione unidirezionale, dall'input agli output, attraversando vari strati senza creare cicli o collegamenti inversi.

La struttura organizzativa delle reti feed-forward è articolata in strati ben definiti, con ciascuno strato composto da un gruppo di neuroni. Questi strati operano in sequenza:

1. Ciascun neurone riceve input dallo strato precedente.
2. Elabora le informazioni ricevute.
3. Inoltra l'informazione elaborata allo strato successivo.

La divisione in strati si articola in tre categorie principali: gli strati di input che accolgono i dati iniziali, gli strati nascosti che li elaborano, e gli strati di output che forniscono il risultato finale del processo. Gli strati nascosti (*hidden layers*) sono particolarmente significativi, poiché permettono alla rete di effettuare calcoli complessi e di rivelare relazioni non immediatamente evidenti nei dati.

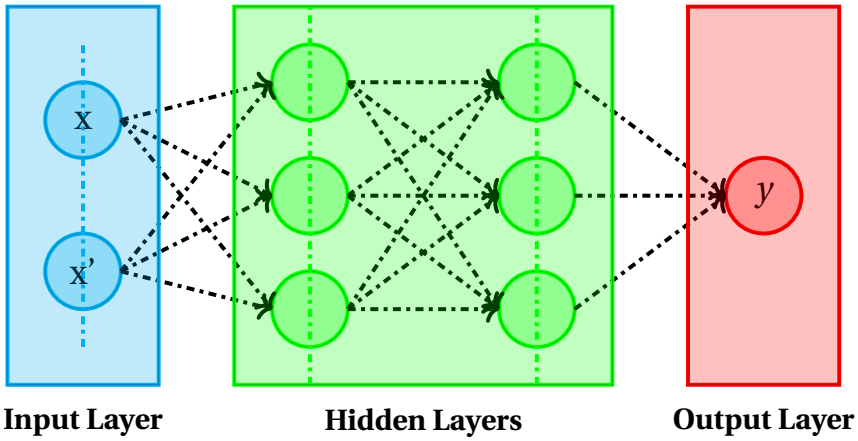


Figure 3.9: Neural Network - *Feed-Forward*

Ogni neurone in uno strato nascosto somma i suoi input ponderati (pesi - ϕ) e applica una funzione di attivazione, che può essere lineare o non lineare, come la sigmoid, la tanh o la ReLU (*Rectified Linear Unit*). Queste funzioni di attivazione aiutano a introdurre la non linearità nel modello, permettendo alle reti di apprendere e modellare relazioni complesse nei dati.

Le reti neurali *feed-forward* vengono addestrate associando input specifici ad output desiderati, puntando a minimizzare la discrepanza tra le previsioni della rete e gli output reali.

Le applicazioni delle reti neurali feed-forward spaziano in diversi ambiti. Vengono impiegate per classificare input in categorie specifiche, come nella classificazione di immagini o nell'identificazione di categorie testuali. Sono utili anche in problemi di regressione, dove è necessario prevedere valori continui, e nel riconoscimento di pattern, dove possono identificare schemi o strutture complesse nei dati.

Politecnico di Torino

3.3.2 CNNs e RNNs

Le Convolutional Neural Networks (CNNs) e le Recurrent Neural Networks (RNNs) sono le due delle architetture più avanzate e influenti, ciascuna con caratteristiche uniche che le rendono particolarmente adatte a compiti specifici e tipologie di dati.

Convolutional Neural Networks

Le Reti Neurali Convoluzionali (CNN) sono state progettate per imitare il modo in cui il cervello umano elabora le informazioni visive. Queste reti sono composte da diversi tipi di strati che lavorano in sinergia per estrarre e interpretare le caratteristiche dalle immagini. I componenti chiave di una CNN includono strati convoluzionali, strati di pooling, e strati completamente connessi.

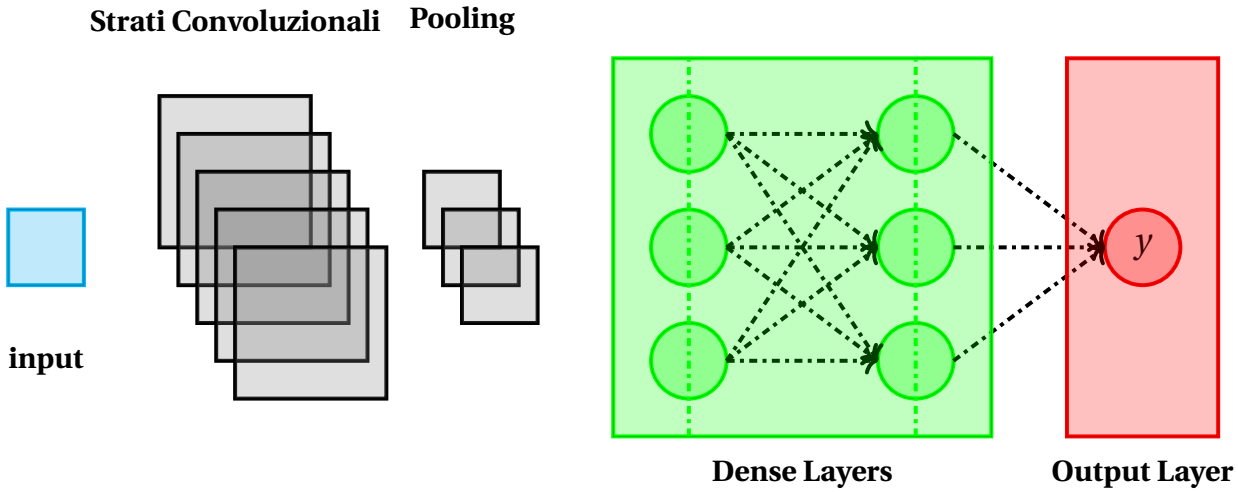


Figure 3.10: *Convolutional Neural Networks*

1. **Strati Convoluzionali**: Il cuore delle CNN è lo strato convoluzionale, che utilizza filtri convoluzionali per catturare le caratteristiche spaziali delle immagini. Ogni filtro si concentra su caratteristiche locali, come bordi o texture, e si sposta sull'intera immagine, creando una mappa delle caratteristiche. Questo processo aiuta la rete a rilevare le stesse caratteristiche in diverse posizioni dell'immagine, conferendo alle CNN una certa invarianza traslazionale.
2. **Strati di Pooling**: Seguono gli strati convoluzionali e servono a ridurre la dimensione spaziale delle mappe delle caratteristiche, comprimendo l'informazione e riducendo la complessità computazionale. Il pooling più comune è il max pooling, che seleziona il valore massimo da una regione della mappa delle caratteristiche, preservando le caratteristiche più prominenti.
3. **Strati *Fully Connected***: Verso la fine dell'architettura della CNN, gli strati completamente connessi utilizzano le caratteristiche estratte dagli strati precedenti per classificare l'immagine in categorie. Questi strati funzionano come una rete neurale tradizionale, dove ogni nodo è connesso a tutti i nodi dello strato precedente.

Politecnico di Torino

Recurrent Neural Networks

A differenza delle reti neurali *feed-forward*, le RNN hanno la distintiva capacità di mantenere uno stato interno o memoria, che consente loro di elaborare e memorizzare informazioni in sequenze di dati. Questa caratteristica le rende ideali per applicazioni che richiedono la comprensione del contesto temporale o sequenziale, come il linguaggio naturale o le serie temporali.

Il componente chiave che distingue le RNN è la presenza di cicli all'interno della rete. In una RNN, l'output di un neurone può essere reinviato allo stesso neurone come parte dell'input per lo step temporale successivo.

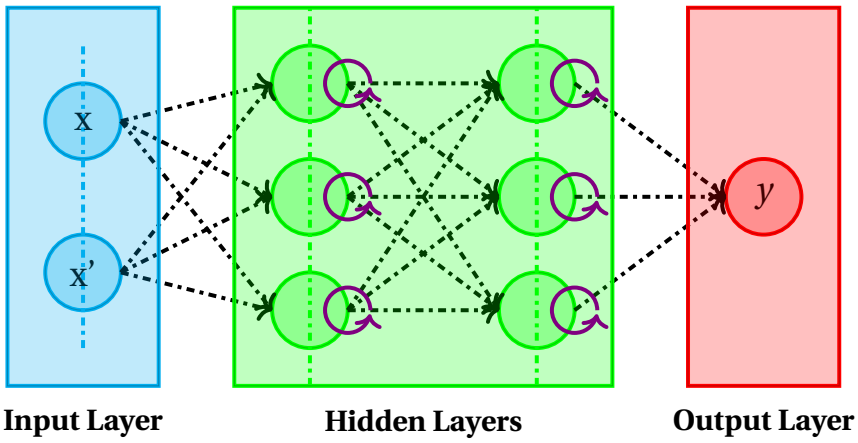


Figure 3.11: *Recurrent Neural Networks*

L'*hidden layer* che viene trasmesso da un passo temporale al successivo, funge da memoria della rete, mantenendo traccia delle informazioni rilevanti elaborate fino a quel momento. L'aggiornamento dello stato nascosto e l'output della rete a ogni passo temporale dipendono dall'input corrente e dallo stato nascosto precedente, consentendo alla rete di adattare le sue risposte in base al contesto accumulato.

Le RNN utilizzano funzioni di attivazione ricorrenti per processare gli input e aggiornare lo stato nascosto. La funzione di attivazione più comune nelle RNN tradizionali è la tangente iperbolica (tanh), contribuendo a regolare il flusso di informazioni nella rete.

Politecnico di Torino

3.3.3 Funzioni di Attivazione

Le funzioni di attivazione rappresentano un elemento essenziale nel design delle reti neurali, essendo responsabili dell'introduzione di non linearità nel processo di apprendimento. Infatti queste funzioni permettono alle reti neurali di apprendere e rappresentare relazioni complesse tra i dati. Senza l'aggiunta di non linearità tramite queste funzioni, anche le reti neurali più profonde non sarebbero in grado di modellare efficacemente i dati, limitandosi a comportarsi come semplici modelli lineari.

1. **Sigmoid**

   La funzione Sigmoid, che mappa i valori di input in un intervallo tra 0 e 1, è stata una delle prime ad essere utilizzata. Tuttavia, può soffrire di quello che viene chiamato *vanishing gradient*, un fenomeno che rende difficile l'addestramento di reti profonde.

$$a[z] = \sigma[z] = \frac{1}{1+e^{-z}} \qquad e \simeq 2.71828 \qquad \text{Sigmoid} \tag{3.3}$$

2. **Tangente Iperbolica** (Tanh)

   La Tangente Iperbolica, simile alla Sigmoid ma con un intervallo di output che va da -1 a 1, offre una migliore interpretazione dei valori di output negativi, ma non risolve completamente il problema del *vanishing gradient*.

$$a[z] = tanh[z] = \frac{e^{z} - e^{z}}{e^{z} + e^{-z}} \qquad \text{Tangente Iperbolica} \tag{3.4}$$

3. **ReLU** (*Rectified Linear Unit*)

   Negli ultimi anni, la ReLU è diventata la funzione di attivazione più popolare, in particolare nelle reti neurali profonde, grazie alla sua semplicità computazionale e alla capacità di mitigare il problema del *vanishing gradient*. La ReLU ha l'effetto di annullare tutti i valori negativi dell'input, mentre mantiene invariati i valori positivi. Questa caratteristica la rende particolarmente efficace nel preservare e propagare i gradienti attraverso la rete durante l'addestramento.

$$a[z] = ReLU[z] = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases} \qquad \text{Rectified Linear Unit} \tag{3.5}$$

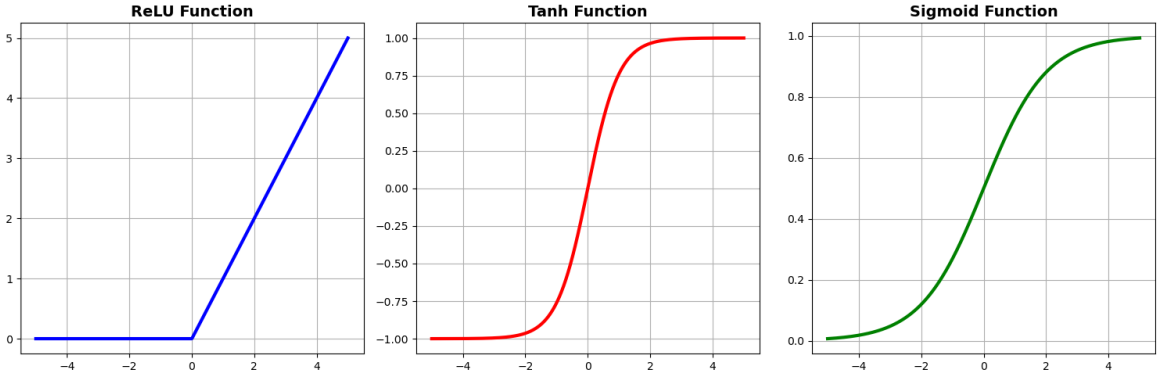


Figure 3.12: Funzioni di attivazione di una NN a confronto.

Politecnico di Torino

La funzione di attivazione ReLU, nonostante sia prediletta dai ricercatori per la sua generalità di utilizzo, è soggetta al problema noto come *dying ReLU problem*, ovvero una forma del problema più generale del *vanishing gradient*. Questa problematica si manifesta poiché la derivata della funzione si annulla per input negativi, portando a un arresto dell'apprendimento dei parametri del modello durante il *training*, specialmente se una porzione significativa del dataset presenta valori negativi.

Per mitigare il *dying ReLU problem*, sono state introdotte delle nuove funzioni di attivazione, tra cui la funzione di attivazione *Swish*, caratterizzata da un parametro β, definito come *learned parameter*. La funzione Swish è descritta dalla seguente espressione matematica:

$$a[z] = Swish[z] = \frac{z}{\left(1+\exp(-\beta x)\right)} \qquad \text{Swish} \tag{3.6}$$

Inoltre, è stata proposta una sua approssimazione, nota come *HardSwish*, che semplifica ulteriormente il calcolo:

$$a[z] = HardSwish[z] = \begin{cases} 0 & z < -3 \\ \dfrac{z(z+3)}{6} & -3 \leq z \leq 3 \\ z & z > 3 \end{cases} \tag{3.7}$$

La selezione della funzione di attivazione ottimale è influenzata da diversi fattori, inclusi la specificità del problema da risolvere, l'architettura della rete neurale e la tipologia dei dati processati.

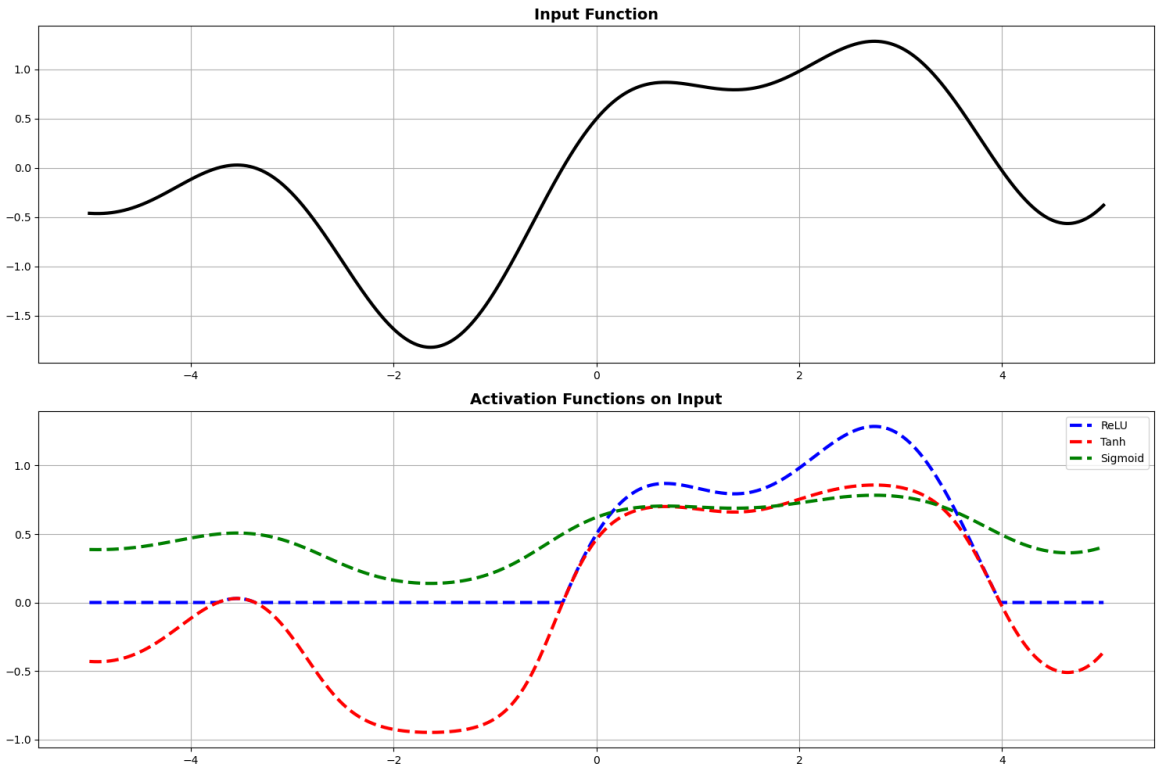


Figure 3.13: Comportamento delle funzioni di attivazioni ad un una funzione di input.

Politecnico di Torino

3.3.4 Loss Function

Come abbiamo già accennato, una rete neurale (NN) apprende dai suoi errori, misurando la discrepanza tra l'output predetto Y' e il valore reale Y. Questa misurazione è affidata alla *Loss Function* $L[\phi]$, nota anche come *cost function* o *objective function*. La funzione di perdita, all'interno dell'architettura dei modelli di Deep Learning (DL), ha il compito di fornire un valore numerico scalare che rappresenti la discrepanza tra il valore predetto dalla rete e il valore reale presente nel dataset.

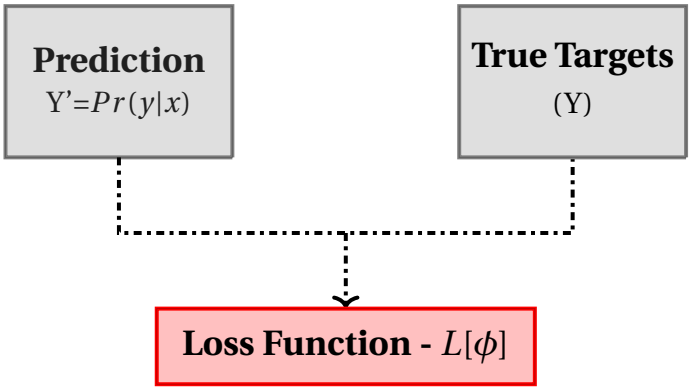


Figure 3.14: Loss Function di un modello di DL.

Cambiando prospettiva, possiamo considerare i modelli non più solo come strumenti per calcolare direttamente una previsione Y, ma piuttosto come mezzi per calcolare una distribuzione di probabilità condizionata[8] $Pr(y|x)$ sugli output possibili y, dati gli input x.

Per costruire una distribuzione su questi output, si opta per una distribuzione parametrica[9] $Pr(y|\theta)$, definita sul dominio di output y. La rete neurale viene quindi impiegata per calcolare uno o più parametri θ di questa distribuzione.

$$\hat{\phi} = \underset{\phi}{arg\,max}\Big[\prod_{i=1}^{I} Pr(y_i|f[x_i,\phi])\Big] \qquad \begin{matrix}\text{Maximum}\\ \text{Likelihood Criterion}\end{matrix} \tag{3.8}$$

Il calcolo di questi parametri si avvale del criterio di massima verosimiglianza (*maximum likelihood*), che seleziona i parametri del modello ϕ in modo da massimizzare la probabilità complessiva su tutti gli esempi di addestramento. Tuttavia, il calcolo diretto della massima verosimiglianza può risultare impraticabile a causa della piccola dimensione di ciascun termine $Pr(y_i|f[x_i,\phi])$, che rende il prodotto di molti di questi termini estremamente ridotto.

[8] Una distribuzione di probabilità condizionata è una misura che descrive la probabilità di un evento o il comportamento di una variabile casuale quando è presente una determinata condizione o informazione.

[9] Una distribuzione parametrica è un tipo di distribuzione di probabilità che può essere completamente definita da un insieme finito di parametri. Questi parametri catturano le caratteristiche essenziali della distribuzione, come la media, la varianza, la forma, e così via.

$$\theta = \mu, \sigma^2 \qquad \begin{cases}\mu: & \text{Media}\\ \sigma: & \text{Varianza}\end{cases}$$

Politecnico di Torino

Una soluzione pratica consiste nel massimizzare il logaritmo della verosimiglianza, convertendo il prodotto in una somma:

$$\log L(\phi) = \sum_{i=1}^{N} \log Pr(y_i | f[x_i, \phi]) \qquad \begin{matrix}\text{Maximum} \\ \text{Log-Likelihood}\end{matrix} \tag{3.9}$$

Questo approccio, noto come criterio di massimizzazione della log-verosimiglianza, mantiene l'equivalenza con il criterio di massima verosimiglianza originale, grazie alla natura monotona crescente della funzione logaritmica.

Inoltre, la massimizzazione della log-verosimiglianza presenta il vantaggio pratico di utilizzare una somma di termini anziché un prodotto, semplificando la computazione con aritmetica a precisione finita.

Per allinearsi alla convenzione di modellare i problemi di adattamento del modello in termini di minimizzazione di una perdita, il criterio di massima log-verosimiglianza viene trasformato in un problema di minimizzazione moltiplicando per -1, ottenendo così il criterio di minimizzazione della log-verosimiglianza negativa:

$$L(\phi) = -\log L(\phi) = -\sum_{i=1}^{N} \log Pr(y_i | f[x_i, \phi]) \qquad \begin{matrix}\text{Minimizing Negative} \\ \text{Log-Likelihood}\end{matrix} \tag{3.10}$$

Questa trasformazione dà origine alla funzione di perdita finale $L[\phi]$, che viene minimizzata durante l'addestramento del modello.

Inoltre, sebbene la rete neurale determini una distribuzione di probabilità su y, spesso, durante l'inferenza[10], si desidera ottenere una stima puntuale piuttosto che una distribuzione. In questi casi, si restituisce il valore di massimo della distribuzione, che corrisponde ai parametri della distribuzione θ previsti dal modello. Ad esempio, nella distribuzione normale, il massimo si verifica alla media μ.

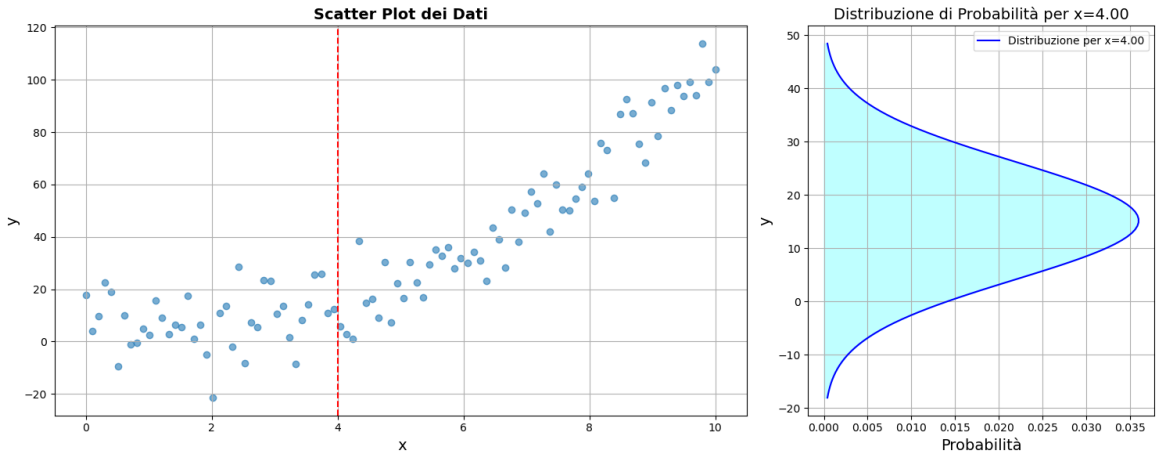


Figure 3.15: Esempio di una distribuzione di probabilità considerando un punto nel dominio dei dati.

[10] L'inferenza in un modello di Deep Learning è il processo attraverso il quale il modello fornisce un output in risposta a nuovi dati di input.

Politecnico di Torino

In sintesi, la procedura per costruire funzioni di perdita per i dati di addestramento, seguendo l'approccio di massima verosimiglianza, è la seguente:

1. Scegliere una distribuzione di probabilità appropriata $Pr(y|\theta)$, definita sul dominio delle previsioni y, con parametri di distribuzione θ.
2. Configurare il modello di machine learning $f[x,\phi]$ per prevedere uno o più di questi parametri, in modo che $\theta = f[x,\phi]$ e $Pr(y|\theta) = Pr(y|f[x,\phi])$.
3. Determinare i parametri della rete $\hat{\phi}$ che minimizzino la funzione di perdita di log-verosimiglianza negativa sul set di dati di addestramento.

Per problemi di regressione, dove l'obiettivo è prevedere valori continui, la *Mean Squared Error* (MSE) è spesso scelta per la sua semplicità e efficacia. Considerando n osservazioni la formulazione matematica della MSE è la seguente:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y_i')^2 \qquad \text{Mean Squared Error} \tag{3.11}$$

Nel contesto della classificazione, la *Cross-Entropy Loss* (o log loss) è comunemente utilizzata, specialmente per problemi di classificazione binaria o multi-classe.

$$CEL = -\sum_{i}^{n} y_i \log(p_1) \qquad \text{Cross-Entropy Loss} \tag{3.12}$$

- **Classificazione Binaria**: La cross-entropy loss per la classificazione binaria è data da:

$$BCEL = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(p_i) + (1-y_i)\log(1-p_i)] \qquad \text{Binary CEL} \tag{3.13}$$

  Dove y_i è il valore reale binario (0 o 1) e p_i è la probabilità predetta, ovvero l'output della NN, che l'osservazione appartenga alla classe 1.

- **Classificazione Multiclasse**: Per la classificazione multiclasse, è utilizzata la *Categorical Cross-Entropy Loss* e si calcola come:

$$CCEL = -\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(p_{ij}) \qquad \text{Categorical CEL} \tag{3.14}$$

  Dove m è il numero di classi, y_{ij} è 1 se l'osservazione i appartiene alla classe j, e p_{ij} è la probabilità predetta che l'osservazione i appartenga alla classe j.

Il punteggio di perdita (*loss score*) calcolato dalla *Loss Function* orienta il modello verso aggiornamenti mirati dei parametri. Tramite l'algoritmo di *backpropagation* e l'uso di un *optimizer*, il modello lavora iterativamente per ridurre questo punteggio di perdita, affinando le sue predizioni ad ogni passo.

La scelta della Loss Function ha un impatto significativo sul modo in cui la rete apprende. Una funzione di perdita mal scelta può portare a risultati inaccurati o ad una convergenza molto lenta durante l'addestramento. Pertanto, è essenziale selezionare una Loss Function che si adatti bene alla natura del problema specifico e ai dati disponibili.

Politecnico di Torino

3.3.5 Optimizer

L'efficacia delle neural networks risiede nella loro capacità di utilizzare il *loss score* come feedback per modificare i pesi dei neuroni. L'ottimizzatore attua questa modifica, ad esempio tramite il metodo del *gradient descent*, ottimizzando i pesi per minimizzare il *loss score*.

Gli ottimizzatori determinano, in base al calcolo del gradiente della funzione di perdita, come e quando i pesi della rete vengono aggiornati durante il processo di addestramento. Il gradiente indica la direzione nella quale la funzione di perdita cresce più rapidamente. L'obiettivo dell'ottimizzatore è quello di muoversi nella direzione opposta al gradiente per ridurre la perdita.

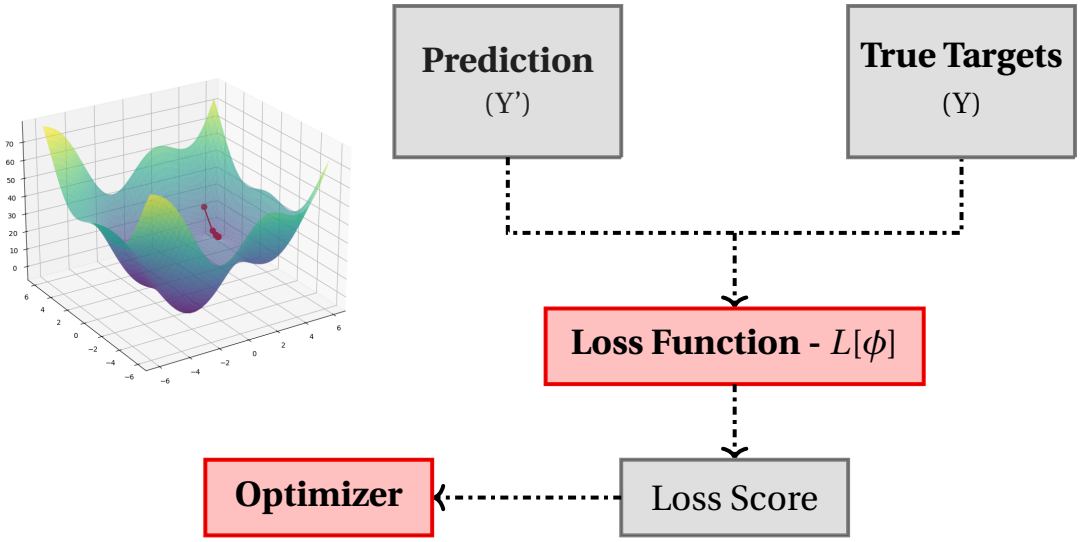


Figure 3.16: Optimizer di una Neurla Network

La scelta dell'ottimizzatore può avere un impatto significativo sulle prestazioni del modello, sulla velocità di convergenza e sulla capacità di evitare minimi locali. Gli ottimizzatori operano nel dominio dell'ottimizzazione non lineare, dove l'obiettivo è trovare il minimo di una funzione complessa, in questo caso, la *loss function*.

Il processo di ottimizzazione in una rete neurale si basa sul calcolo del gradiente, che è il vettore delle derivate parziali della funzione di perdita rispetto a ciascun peso della rete.

$$\nabla L(\phi) = \left(\frac{\partial L}{\partial \phi_1}, \frac{\partial L}{\partial \phi_2}, \ldots, \frac{\partial L}{\partial \phi_n} \right) \qquad \text{Gradiente della } \textit{Loss Function} \tag{3.15}$$

L'efficacia di un ottimizzatore dipende dalla sua capacità di navigare il paesaggio complesso delle funzioni di perdita, che possono presentare sfide come minimi locali, punti di sella, e plateaux. L'obiettivo è quello di evitare di rimanere intrappolati in minimi locali subottimali e di convergere efficacemente verso un minimo globale, o almeno un minimo locale accettabile.

Politecnico di Torino

Il concetto base del Gradient Descent è aggiornare i pesi della rete neurale in modo da minimizzare la funzione di perdita. L'aggiornamento dei pesi avviene in direzione opposta al gradiente della funzione di perdita, come descritto dalla seguente formula:

$$\phi = \phi - \alpha \nabla L(\phi) \tag{3.16}$$

Dove è stato indicato con α è il tasso di apprendimento (*learning rate*).

Tra gli ottimizzatori più comunumenente utilizzati ritroviamo:

1. **Varianti del Gradient Descent**:

   (a) Una variante del Gradient Descent è lo *Stochastic Gradient Descent* (SGD), che utilizza un singolo esempio di addestramento per l'aggiornamento dei pesi, piuttosto che l'intero set di dati. Questo rende l'SGD più veloce e meno incline a rimanere intrappolato in minimi locali. L'aggiornamento dei pesi in SGD è dato da:

   $$\phi = \phi - \alpha \nabla L\Big(\phi; x_i, y_i\Big) \qquad \text{SGD} \tag{3.17}$$

   (b) Il *Mini-batch Gradient Descent* è un compromesso tra il Gradient Descent batch completo e l'SGD. Utilizza un sottoinsieme (mini-batch) del set di addestramento per ogni aggiornamento, fornendo un equilibrio tra l'efficienza computazionale e la stabilità dell'aggiornamento dei pesi.

2. **Momentum e Nesterov Accelerated Gradient**:

   (a) *Momentum* è un metodo che aiuta ad accelerare l'SGD nella direzione corretta e smorzare le oscillazioni. Incorpora la nozione di inerzia nel processo di aggiornamento dei pesi, considerando i gradienti passati oltre a quello attuale.

   $$\begin{cases} v_{\text{new}} &= \gamma v_{\text{old}} + \alpha \nabla L(\phi) \\ \phi &= \phi - v_{\text{new}} \end{cases} \qquad \text{Momentum} \tag{3.18}$$

   Dove v rappresenta la velocità e γ è il parametro di Momentum.

   (b) Il *Nesterov Accelerated Gradient* (NAG) è una raffinazione del metodo di Momentum. NAG anticipa la posizione futura della *loss function* e compie aggiustamenti più informati dei pesi.

   $$\begin{cases} v_{\text{new}} &= \gamma v_{\text{old}} + \alpha \nabla L(\phi - \gamma v_{\text{old}}) \\ \phi &= \phi - v_{\text{new}} \end{cases} \qquad \text{NAG} \tag{3.19}$$

   Questa formula tiene conto non solo del gradiente attuale, ma anche della direzione e della velocità dei cambiamenti precedenti.

3. **Adaptive Moment Estimation - Adam**:

   Adam si adatta al cambiamento dei gradienti. In altre parole, è capace di modulare il tasso di apprendimento specifico per ciascun parametro della rete neurale, basandosi sull'andamento dell'addestramento. Data la sua robustezza, anche in termini di velocità di convergenza, e versatilità è generalmente l'algoritmo maggiormente utilizzato nell'ambito del DL.

Politecnico di Torino

3.4 ALLENAMENTO DELLE RETI NEURALI

L'efficacia di un modello di Deep Learning è strettamente legata alla qualità del suo allenamento. Un modello ben addestrato può riconoscere schemi complessi nei dati, fare previsioni accurate e generalizzare bene su dati non visti durante l'addestramento. Tuttavia, raggiungere questo livello di prestazione richiede attenzione a diversi aspetti chiave dell'allenamento, tra cui la selezione del dataset, la scelta della *loss function*, l'ottimizzazione dei parametri, e la gestione di fenomeni come l'overfitting.

Dataset

Un buon dataset di addestramento dovrebbe essere rappresentativo del problema da risolvere, ampio abbastanza da coprire tutte le variazioni rilevanti e bilanciato in termini di distribuzione delle classi o delle caratteristiche. Infatti la qualità e la quantità dei dati raccolti influenzano direttamente le prestazioni del modello. Un dataset ampio e diversificato può aiutare il modello a generalizzare meglio e a ridurre il rischio di overfitting.

In contesti di apprendimento supervisionato, i dataset sono tipicamente composti da esempi di input insieme alle loro etichette corrispondenti. Per esempio, in un problema di classificazione di immagini, ogni immagine nel dataset avrà un'etichetta che indica la classe di appartenenza.

Generalmente il dataset viene suddiviso in subsets specifici. Questa suddivisione è fondamentale per valutare l'efficacia del modello in diverse fasi del suo addestramento e per garantire che il modello sia in grado di generalizzare bene su dati non visti durante l'addestramento. Ciascun subset svolge un ruolo distinto e contribuisce a diversi aspetti dell'addestramento e della valutazione del modello.

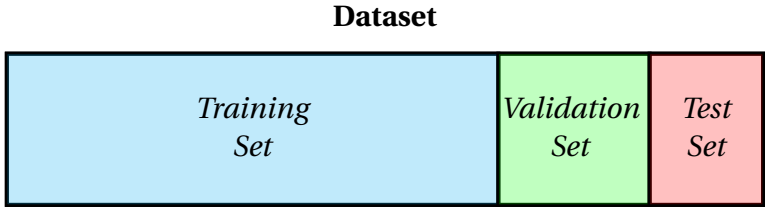


Figure 3.17: Suddivisione del Dataset in: *Training Set*, *Validation Set* e *Test Set*.

1. **Training Set**: Il *Training Set* è la porzione di dataset utilizzata per addestrare il modello. Comprende una vasta gamma di esempi che permettono alla rete neurale di apprendere e adattarsi. Durante la fase di addestramento, la rete neurale modifica i suoi pesi e bias per minimizzare l'errore nelle previsioni basandosi su questi dati. La dimensione e la varietà del Training Set sono cruciali per la capacità del modello di apprendere efficacemente e generalizzare su nuovi dati.

   Una sfida chiave nell'utilizzo del Training Set è evitare l'overfitting. L'overfitting si verifica quando la rete neurale diventa troppo specializzata sui dati di addestramento, perdendo la capacità di generalizzare su dati nuovi e non visti. Per prevenire l'overfitting, è importante avere un Training Set rappresentativo dell'intera distribuzione dei dati che il modello si aspetta di incontrare. Ciò include un'adeguata varietà e bilanciamento delle classi nel caso di problemi di classificazione. Un dataset non bilanciato o troppo limitato può portare a una scarsa performance del modello.

2. **Validation Set**: Il *Validation Set* serve a fornire una valutazione imparziale delle prestazioni del modello durante l'addestramento. Non è utilizzato per l'addestramento effettivo, ma piuttosto per regolare i parametri, come il tasso di apprendimento o la complessità della rete. Il Validation Set, inoltre, aiuta a identificare il fenomeno dell'overfitting.

3. **Test Set**: Il *Test Set* è usato per valutare le prestazioni del modello dopo che l'addestramento è stato completato. Offre un'indicazione di come il modello si comporterà su dati nuovi e non visti. Utilizzare un Test Set separato è essenziale per ottenere una valutazione oggettiva dell'efficacia generale del modello.

È di fondamentale importanza che il *Validation Set* sia distinto dal *Test Set*. Mentre il test set viene utilizzato per una valutazione finale e imparziale, il validation set può essere utilizzato più volte durante il processo di addestramento per guidare le decisioni su come ottimizzare il modello.

Overfitting

Uno dei principali ostacoli nell'allenamento dei modelli di Deep Learning è l'overfitting. Quest'ultimo è un fenomeno che si verifica quando un modello di rete neurale impara non solo le relazioni generali dai dati di addestramento, ma anche i rumori e le anomalie specifici a quel particolare set di dati. Ciò può portare a prestazioni apparentemente eccellenti sui dati di addestramento, ma a prestazioni scadenti sui dati di validazione o di test.

L'overfitting nelle reti neurali può essere attribuito a una varietà di cause, alcune delle quali sono intrinseche alla natura dei dati e del modello, mentre altre sono legate alle scelte fatte durante il processo di addestramento.

Per capire l'overfitting introduciamo delle quantità fondamentali all'interno del processo di allenamento di una rete neurale.

1. **Epochs**: Il numero totale di volte che l'intero set di training viene esaminato. L'aumento del numero di epoche può migliorare le prestazioni del modello fino a un certo punto, dopo il quale può verificarsi *overfitting*.

2. **Batch Size**: Indica il numero di esempi di training processati prima dell'aggiornamento dei pesi. Un batch size[11] più piccolo può aumentare la varianza degli aggiornamenti dei pesi, portando potenzialmente a una convergenza più rapida ma meno stabile.

Una causa primaria dell'overfitting è una complessità eccessiva del modello[12] rispetto alla quantità e varietà dei dati di addestramento disponibili. Quando una rete neurale ha un numero eccessivo di neuroni, diventa capace di memorizzare quasi perfettamente i dati di addestramento, comprese le loro peculiarità e irregolarità, piuttosto che apprendere modelli generalizzabili. Questo si verifica soprattutto in scenari dove il numero di parametri della rete è circa dello stesso ordine di grandezza del numero di esempi nel training set.

[11] Generalmente si utilizza una batch size pari a 128 o a dei suoi multipli.

[12] Questa concetto è stato in parte smentito dalla recente scoperta del *double descent*, che mette in chiaro il comportamento dell'overfitting in funzione degli parametri del modello.

Politecnico di Torino

Un'altra causa comune è la durata eccessiva dell'addestramento. Se una rete neurale viene addestrata per troppo tempo (troppe *epochs*), può iniziare a ridurre la *loss function* sui dati di addestramento a scapito della performance sui dati nuovi o non visti. Questo è particolarmente vero in assenza di tecniche come l'*early stopping*, che sono progettate per prevenire l'addestramento eccessivo.

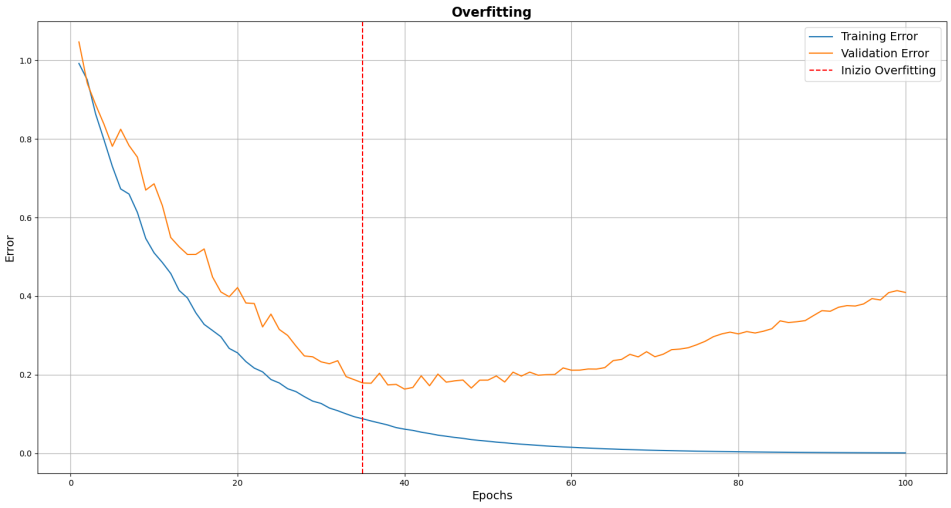


Figure 3.18: Overfitting

Un segnale chiaro di overfitting si verifica quando c'è una significativa discrepanza tra la performance del modello sui dati di addestramento e quella sui dati di validazione. In termini più specifici, l'overfitting è spesso indicato da una bassa funzione di perdita durante l'addestramento, ma una funzione di perdita elevata o crescente sui dati di validazione. Questa situazione implica che il modello sta apprendendo le specificità dei dati di addestramento, piuttosto che estrarre schemi generali applicabili a nuovi dati.

La principale conseguenza dell'overfitting è una riduzione della capacità di generalizzazione del modello. Questo si manifesta in una performance notevolmente inferiore su set di dati esterni rispetto a quelli ottenuti durante l'addestramento. In pratica, ciò significa che, nonostante il modello possa apparire altamente accurato durante la fase di addestramento, la sua utilità pratica è limitata, in quanto non può fare previsioni affidabili su nuove istanze di dati.

Inoltre, l'overfitting può portare a conclusioni errate sull'importanza di alcune caratteristiche dei dati. Un modello overfitted potrebbe dare peso eccessivo a caratteristiche irrilevanti o casuali trovate nel training set, portando a un'interpretazione distorta delle relazioni sottostanti nei dati. Questo è particolarmente problematico in applicazioni come la diagnostica medica o la previsione finanziaria, dove decisioni errate possono avere conseguenze gravi.

Politecnico di Torino

Double Descent

Inizialmente si credeva che, man mano che la complessità del modello aumentava, la performance sul set di validazione migliorasse fino a un punto di ottimalità, oltre il quale iniziava a peggiorare a causa dell'overfitting. Tuttavia, il *double descent* rivela che dopo un periodo iniziale di overfitting, aumentando ulteriormente la complessità, la performance del modello può iniziare a migliorare nuovamente.

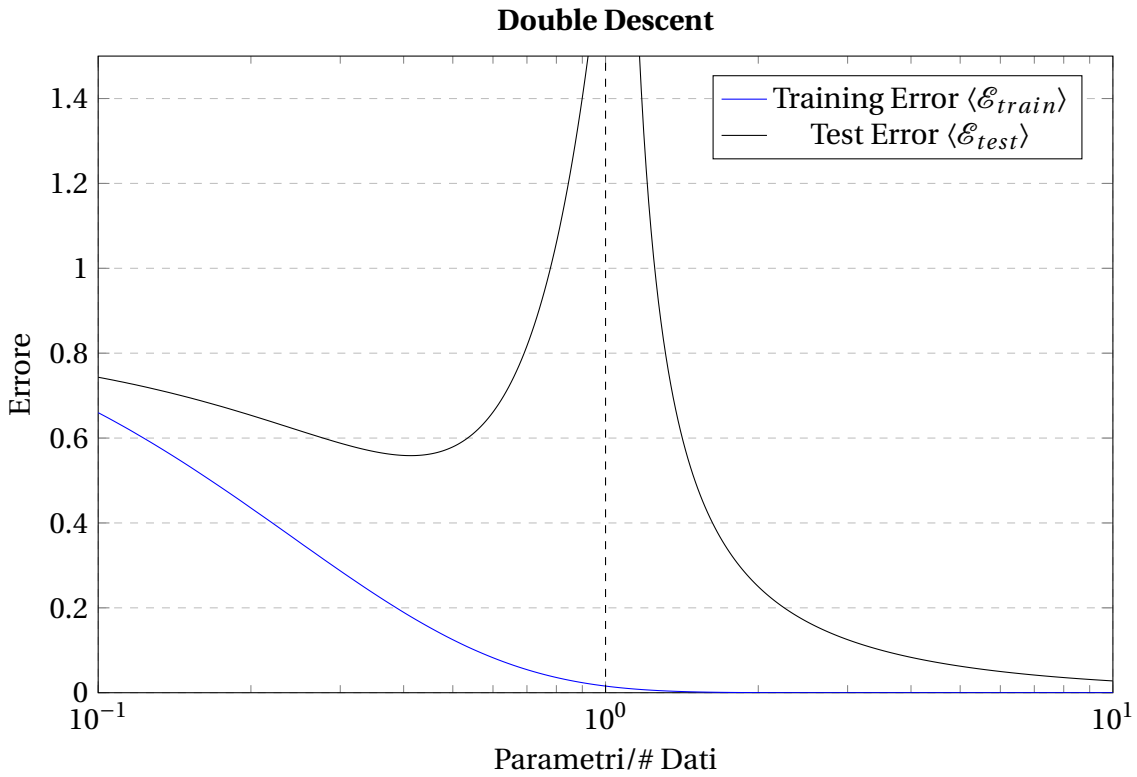


Figure 3.19: Fenomeno del *Double Descent*

Il double descent si verifica in un contesto in cui la complessità del modello di rete neurale supera il numero di punti dati nel training set. In questo scenario, la curva di errore presenta una prima fase di discesa, seguita da un picco (che rappresenta il punto in cui il modello inizia a soffrire di overfitting) e poi da una seconda fase di discesa, che sorprendentemente mostra un miglioramento delle performance del modello.

Questa seconda fase di discesa suggerisce che quando un modello diventa sufficientemente complesso, è in grado di apprendere i rumori dei dati di addestramento in modo tale da migliorare le sue prestazioni generali. In altre parole, il modello non solo apprende le caratteristiche sottostanti dei dati, ma anche le loro specifiche anomalie, e ciò, inaspettatamente, contribuisce a una migliore generalizzazione sui nuovi dati.

Un aspetto fondamentale del double descent è che si manifesta principalmente in reti neurali di grande dimensione, dove il numero di parametri supera significativamente la dimensione del dataset.

Una delle principali implicazioni è la necessità di ripensare la tradizionale comprensione dell'overfitting. Invece di evitare sistematicamente modelli altamente complessi per paura dell'overfitting, il double descent suggerisce che, in certi contesti, incrementare ulteriormente la complessità potrebbe essere benefico.

Politecnico di Torino

CHAPTER 4

REINFORCEMENT LEARNING - RL

Nel panorama evolutivo del Machine Learning (ML), il Reinforcement Learning (RL) si distingue come un paradigma potente e versatile, attingendo a principi di ottimizzazione sequenziale e teoria delle decisioni. Mentre algoritmi di Supervised Learning e Unsupervised Learning esplorano rispettivamente dati etichettati e strutture nascoste, il RL si focalizza sull'interazione dinamica con un ambiente complesso per raggiungere un obiettivo definito.

Gli algoritmi RL, ispirati dai processi cognitivi umani e dall'apprendimento animale, simulano un'entità decisionale, o *agente*, che apprende bilanciando quello che in termini anglosassoni viene chiamato *Exploration-Exploitation trade-off*, ossia un compromesso tra l'esplorazione dell'ambiente e lo sfruttamento delle conoscenze acquisite in precedenza. L'agente interagisce con l'*environment*, eseguendo azioni e osservando le conseguenze, in un ciclo continuo di feedback che raffina le sue strategie decisionali.

Questo processo, guidato dal meccanismo di prova ed errore, permette all'agente di collezionare informazioni critiche sulle conseguenze delle sue azioni. In base a queste informazioni, l'algoritmo di RL modifica il proprio comportamento per ottimizzare un obiettivo, solitamente espresso in termini di ricompense (*Rewards*) e penalità.

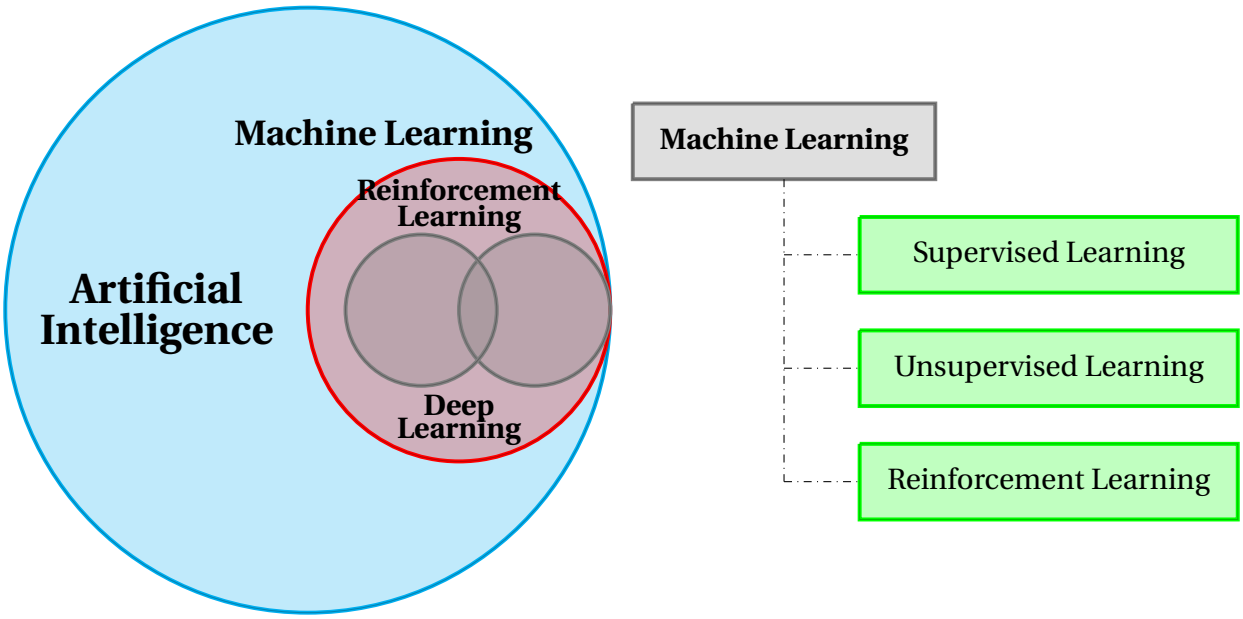


Figure 4.1: AI, Machine Learning and Reinforcement Learning

Il cuore pulsante del RL è l'intenzione di massimizzare un valore numerico complessivo, il *Return*. L'agente, ad ogni azione, riceve una ricompensa numerica, il *Reward*, che serve da indicatore per la bontà dell'azione compiuta rispetto agli obiettivi prefissati. Questo feedback immediato, unito all'esperienza accumulata, guida l'adattamento delle strategie future.

Ogni decisione presa dall'agente ha un impatto sia immediato che a lungo termine: un'azione al tempo t (s_t) non solo determina lo stato successivo al tempo $t+1$ (s_{t+1}), ma plasma anche la traiettoria futura degli eventi. Questo principio di *Delayed Reward*, che riflette le dinamiche della realtà, è cruciale: l'agente deve valutare i benefici immediati in relazione agli effetti futuri, spesso incerti e complessi.

Inoltre, il RL trova applicazione in una varietà di domini complessi, da giochi strategici come gli scacchi e Go[1], a sfide più pratiche e quotidiane come la navigazione autonoma e la gestione delle risorse. In ogni contesto, l'agente RL apprende a navigare in un ambiente pieno di incertezze e dinamiche variabili, aspirando a decisioni che massimizzano il *Return* a lungo termine.

Nel campo dei giochi, il RL ha dimostrato capacità sorprendenti. Programmi come AlphaGo e il suo successori, AlphaGo-Zero e AlphaZero, hanno dimostrato che gli algoritmi RL possono non solo imparare giochi complessi ma anche sviluppare strategie innovative e superiori a quelle umane. Questi successi non sono limitati a giochi tradizionali, il RL viene utilizzato anche in videogiochi moderni e simulazioni, dove l'agente deve reagire a scenari dinamici e imprevedibili, spesso superando le sfide poste da avversari umani.

Oltre ai giochi, il RL trova impiego significativo nella robotica. Gli agenti RL sono addestrati per svolgere compiti come la manipolazione di oggetti, la navigazione in spazi complicati e la collaborazione con altri robot o esseri umani. In questi scenari, il RL aiuta i robot a capire come ottimizzare le sequenze di movimenti e decisioni per completare compiti con efficienza e precisione crescenti.

Nonostante queste applicazioni promettenti, il RL presenta delle sfide. L'apprendimento efficiente in ambienti complessi e dinamici richiede enormi quantità di dati e computazione. Inoltre, la progettazione di sistemi di ricompensa adeguati che guidino efficacemente l'apprendimento dell'agente verso obiettivi desiderati rimane una sfida significativa.

Il meccanismo di prova ed errore, insieme al principio di *Delayed Reward* e all'accento sulle conseguenze a lungo termine delle azioni, non solo distingue il RL dagli altri paradigmi di apprendimento ma apre anche nuove prospettive per l'autonomia e l'efficienza degli algoritmi intelligenti.

4.1 Element of Reinforcement Learning

Nel contesto del RL, l'agente è un decisore autonomo che cerca di massimizzare una nozione cumulativa di ricompensa attraverso un processo di scoperta e adattamento. L'apprendimento nel RL è intrinsecamente legato a un insieme di componenti fondamentali che insieme definiscono la sua architettura e le sue capacità.

Consideriamo adesso i pilastri centrali del Reinforcement Learning: l'agente, le value function, la policy, e il meccanismo di ricompensa. Ognuno di questi componenti gioca un ruolo vitale nel processo di apprendimento dell'agente, contribuendo a plasmare le sue decisioni e strategie.

[1] AlphaGo è un noto algoritmo di RL che, nel 2016 ha sconfitto il campione mondiale di Go Lee Se-dol, dimostrando le capacità avanzate del RL nei giochi strategici.

Politecnico di Torino

Agent	L'agente nel Reinforcement Learning è l'entità incaricata di selezionare le azioni, basandosi su una politica (policy) volta a massimizzare il *Return*. In sostanza, l'agente mira a raggiungere l'obiettivo del problema, operando anche in assenza di una conoscenza completa dell'ambiente e delle sue dinamiche. Nel corso delle interazioni, l'agente apprende e perfeziona la sua policy, migliorando progressivamente la sua capacità decisionale e di adattamento all'ambiente sconosciuto o parzialmente noto.
Value Function	Si definiscono due valori, fondamentali per l'ottimizzazione della policy, ovvero la *State Value* e l'*Action Value*: 1. **State Value -** V_s: Rappresenta l'expected return, ovvero la sommatoria dei futuri rewards accumulati dall'agente, partendo da uno stato s e seguendo una determinata policy π. Questo valore indica quanto sia vantaggioso trovarsi in uno specifico stato, data la policy. 2. **Action Value -** $Q(s,a)$: Indica il return, ovvero la sommatoria di tutti i futuri reward lungo l'episodio, che l'agente accumulerà partendo da un determinato stato s e compiendo un'azione a, uguale o diversa dall'azione che avrebbe scelto l'agente sotto la policy π. Successivamente, allo stato $s+1$ l'agente tornerà a scegliere le azioni dettate dalla policy π. Questo valore valuta l'efficacia di eseguire una specifica azione in uno stato dato, considerando le ricompense future ottenibili seguendo la policy. Queste *Value Functions* sono fondamentali per valutare l'efficacia a lungo termine della policy. Per questo motivo, l'agente tende a scegliere le azioni basandosi non solo sul reward immediato ma anche sul loro valore previsto in termini di return futuro, ottimizzando così la policy nel suo complesso.
Policy - π	La policy rappresenta il comportamento dell'agente, ovvero il complesso delle regole o strategie che determinano come l'agente sceglie una particolare azione in risposta a un determinato stato. In sostanza, la policy guida le decisioni dell'agente, stabilendo la probabilità con cui selezionare ogni possibile azione in ogni stato, e viene continuamente adattata e ottimizzata attraverso il processo di apprendimento dell'agente.
Reward - R	Il reward è un valore numerico assegnato all'agente in risposta alle sue azioni, riflesso dell'efficacia o delle conseguenze di quella specifica azione. La definizione e l'assegnazione del reward sono cruciali negli algoritmi di RL, poiché l'agente è programmato per massimizzare la somma totale dei rewards ricevuti. In sostanza, il reward funge da indicatore per l'agente, specificando cosa ci si aspetta che faccia e in che modo, guidando così l'apprendimento e l'adattamento della sua policy verso comportamenti che generano esiti positivi secondo i criteri stabiliti.

Table 4.1: Elementi chiave degli algoritmi di *Reinforcement Learning*.

Politecnico di Torino

4.1.1 MARKOV DECISION PROCESSES - MDP

Il Processo Decisionale di Markov (MDP) costituisce il fulcro teorico del Reinforcement Learning (RL), fornendo un quadro formale per la comprensione dell'apprendimento e della decisione in ambienti dinamici. In questo contesto, l'MDP modella l'interazione tra un agente e il suo ambiente attraverso una serie di decisioni sequenziali, ognuna delle quali influisce non solo sulle ricompense immediate ma anche sulle traiettorie future e sulle ricompense cumulative.

Essenzialmente, un MDP fornisce una struttura matematica per modellare la decisione in situazioni dove gli esiti sono parzialmente casuali e parzialmente sotto il controllo di un decisore. È definito da una serie di elementi chiave che insieme descrivono l'ambiente di decisione:

1. **Stati (States)** - S: Un insieme di stati che descrive tutte le possibili configurazioni dell'agente e dell'ambiente. Ogni stato rappresenta una fotografia univoca del processo decisionale in un dato istante.

2. **Azioni (Actions)** - A: Un insieme di azioni disponibili all'agente. Le azioni rappresentano le scelte che l'agente può fare per influenzare lo stato dell'ambiente.

3. **Funzione di Transizione (Transition Function)** - P: Una funzione che determina la probabilità di passaggio allo stato s', quando l'agente intraprende l'azione (a) nello stato (s). Questa funzione di probabilità incarna la dinamica dell'ambiente, incorporando elementi di casualità e incertezza.

$$p(s', r|s, a) \doteq Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \quad s', s \in S \qquad r \in R \qquad a \in A(s) \tag{4.1}$$

   La distribuzione di probabilità è definita dalle dinamiche dell'ambiente e il passaggio a ogni possibile nuovo stato e il relativo reward sono determinati unicamente dallo stato attuale e dall'azione intrapresa.

4. **Ricompense (Rewards)** - R: Una funzione che assegna un valore numerico come ricompensa all'agente a seguito delle sue azioni. La ricompensa può dipendere dallo stato corrente, dall'azione eseguita e dallo stato successivo, e serve come segnale per guidare l'apprendimento dell'agente verso comportamenti desiderabili.

Il MDP opera quindi all'interno di un insieme definito e comprensivo di stati, rappresentando ogni possibile condizione o configurazione che l'agente potrebbe sperimentare[2]. Questo insieme, noto come spazio degli stati, è fondamentale perché fornisce una rappresentazione completa e chiusa di tutti gli scenari che l'agente deve considerare. In ogni istante, l'agente si trova in uno specifico stato all'interno di questo insieme, che riflette la situazione attuale del sistema o dell'ambiente.

La proprietà fondamentale che rende un processo decisionale un MDP è la cosiddetta "proprietà di Markov". Questa proprietà afferma che la probabilità di transizione verso il prossimo stato dipende solo dallo stato attuale e dall'azione intrapresa, e non da come si è arrivati a tale stato. In altre parole, il futuro è indipendente dal passato, dato il presente. Questa proprietà semplifica notevolmente l'analisi e la computazione nel RL, poiché riduce la complessità del problema eliminando la necessità di considerare l'intera storia delle azioni e degli stati precedenti.

[2] In questo modello matematico si assume che qualsiasi situazione incontrata durante il problema reale possa essere rappresentata da almeno uno stato tra il set dei possibili stati.

Interfaccia Agente-Ambiente

L'interfaccia agente-ambiente rappresenta, nei MDPs, la cornice entro cui l'agente e l'ambiente interagiscono, definendo le regole e la struttura di questa interazione fondamentale. Questa interfaccia stabilisce il modo in cui l'agente percepisce il suo ambiente e come può agire su di esso per raggiungere i suoi obiettivi.

Essa delinea il ciclo di feedback in cui l'agente sperimenta, impara e si adatta. Ogni azione intrapresa dall'agente non solo determina la sua esperienza immediata sotto forma di ricompense, ma anche le sue esperienze future, influenzando così il suo percorso di apprendimento nel tempo.

Dinamiche dell'Interazione

L'interazione tra l'agente e l'ambiente si svolge in una sequenza di eventi iterativi e discreti, i quali si susseguono in intervalli temporali ben definiti. Questi eventi possono essere descritti come segue:

1. **Osservazione dello Stato (State Observation)**: In ogni intervallo temporale, l'agente riceve informazioni sullo stato corrente dell'ambiente, denotato con $S_t \in S$. Questo stato rappresenta la base di conoscenza dell'agente su cui verranno prese le decisioni future. È un punto di riferimento cruciale che riflette la situazione attuale e fornisce il contesto per l'azione successiva.
2. **Selezione dell'Azione (Action Selection)**: Basandosi sullo stato osservato e sulla policy attualmente adottata, l'agente seleziona un'azione, indicata con $A_t \in A$. Questa scelta è il risultato della strategia di decisione dell'agente, che può essere influenzata dalla sua esperienza passata, dalla sua conoscenza dell'ambiente, e dagli obiettivi che mira a raggiungere.
3. **Esecuzione e Feedback**: L'azione selezionata viene eseguita, e l'ambiente reagisce a questa azione cambiando il suo stato a S_{t+1} e fornendo all'agente una ricompensa numerica $R_{t+1} \in R \subset \mathbb{R}$. Questa ricompensa funge da feedback immediato per l'agente, segnalando l'efficacia dell'azione intrapresa rispetto agli obiettivi prefissati.

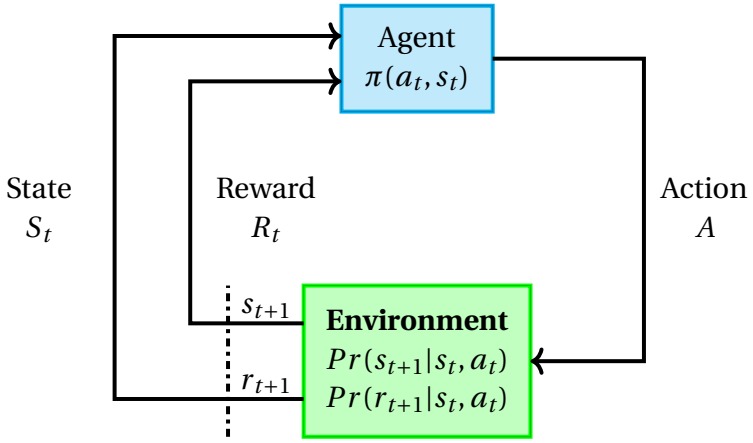


Figure 4.2: Dinamica dell'iterazione tra agente ed ambiente in un MDP [1]

Politecnico di Torino

Questi passaggi si ripetono ciclicamente, formando un ciclo di feedback continuo che si estende per tutta la durata dell'interazione tra agente e ambiente. Ogni sequenza di stati, azioni e ricompense, conosciuta come *traiettoria*, costituisce un percorso unico attraverso lo spazio del problema che l'agente esplora e da cui apprende.

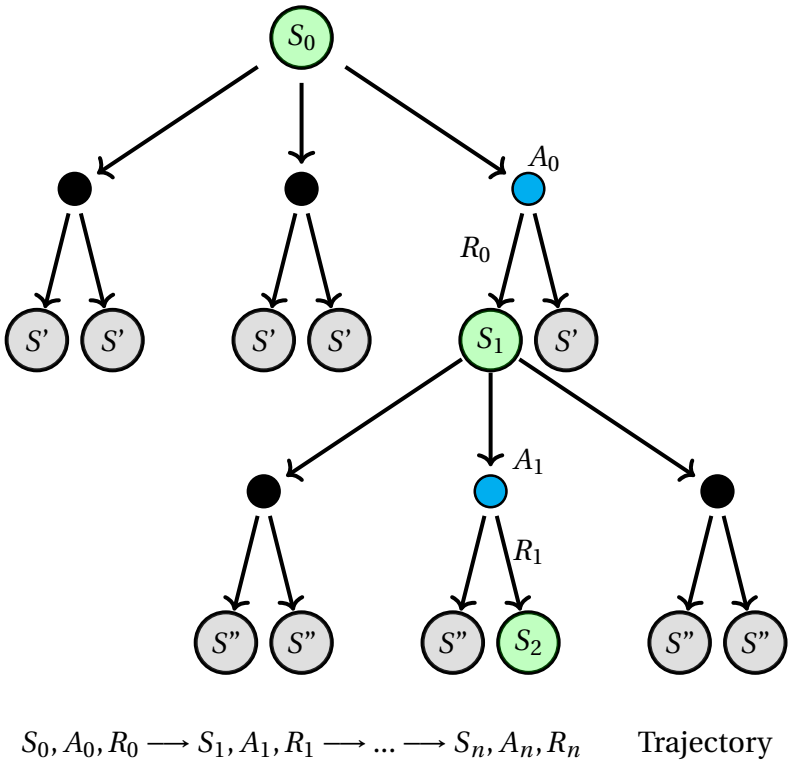


Figure 4.3: Traiettoria compiuta dall'agente attraverso lo spazio del problema.

L'interfaccia agente-ambiente facilita un processo continuativo di apprendimento e decisione. L'agente mira a massimizzare la somma delle ricompense ricevute, apprendendo attraverso un metodo di sperimentazione e errore quali sequenze di azioni conducono ai migliori risultati a lungo termine. Questa interfaccia non solo delinea le regole di interazione tra l'agente e il mondo in cui opera, ma stabilisce anche il contesto in cui avviene l'apprendimento e l'adattamento.

Un vantaggio significativo del Processo Decisionale di Markov (MDP) risiede nella sua struttura astratta e versatile, rendendolo applicabile a una varietà di problemi. In contesti continui, dove il concetto di time-step perde la sua connotazione tradizionale, è possibile reinterpretare questi intervalli non come segmenti fissi nel tempo, ma come momenti decisionali consecutivi che si manifestano durante l'episodio. Questa flessibilità consente di adattare l'MDP a scenari dinamici e non discreti.

Inoltre, l'MDP non impone restrizioni rigide sulle azioni o sugli stati. Le azioni possono essere qualsiasi decisione che si desidera insegnare all'agente, da semplici movimenti a complesse sequenze di operazioni. Gli stati, analogamente, possono rappresentare qualsiasi aspetto dell'ambiente o del sistema che sia rilevante e conoscibile.

La flessibilità dell'interfaccia agente-ambiente e delle dinamiche di interazione nel MDP consente di modellare un'ampia varietà di problemi decisionali. Che si tratti di ambienti discreti o continui, semplici o complessi, il framework fornito dall'MDP offre un potente strumento per comprendere e ottimizzare il processo decisionale e di apprendimento.

Politecnico di Torino

MARKOV REWARD PROCESS

Il Markov Reward Process (MRP) è un'estensione del Processo Decisionale di Markov (MDP) che integra la nozione di ricompensa, un elemento che guida l'agente nel suo processo di apprendimento e decisionale. L'obiettivo primario dell'agente in un MRP è massimizzare la ricompensa accumulata, la quale è strettamente legata alle transizioni di stato e alle traiettorie intraprese durante gli episodi.

La *reward function* riveste un ruolo fondamentale, poiché fornisce all'agente gli indizi necessari per orientare il suo apprendimento verso gli obiettivi desiderati. La progettazione accurata di questa funzione è quindi essenziale per assicurare che l'agente adotti comportamenti che siano in linea con gli obiettivi complessivi del sistema.

Consideriamo due scenari distinti per illustrare come la *reward function* possa influenzare l'apprendimento dell'agente:

1. *Chess Environment*: In un ambiente in cui l'agente deve imparare a giocare a scacchi, è fondamentale progettare una *reward function* che rifletta l'obiettivo finale del gioco: vincere la partita. Se si assegnasse una ricompensa per ogni pezzo catturato, l'agente potrebbe perseguire una strategia miope focalizzata sulla cattura dei pezzi piuttosto che sulla strategia complessiva per vincere la partita. Pertanto, è preferibile assegnare la ricompensa solo al termine della partita, basandola sull'esito finale:

   $r_{end} = +1$ per la Vittoria, $r_{end} = -1$ per la Sconfitta, $r_{end} = 0$ per il Pareggio.

2. *Maze Environment*: In un ambiente dove l'obiettivo dell'agente è uscire il più rapidamente possibile da un labirinto, una *reward function* efficace potrebbe assegnare una penalità (-1) per ogni time step trascorso nel labirinto. Questo incentiva l'agente a trovare il percorso più rapido verso l'uscita, poiché minimizza il numero di penalità (ricompense negative) ricevute

Questi esempi evidenziano come la *reward function* debba essere attentamente progettata per riflettere gli obiettivi desiderati e guidare l'agente verso comportamenti ottimali. La definizione di ricompensa segue la **Reward Hypothesis**, che afferma:

> *That all of what we mean by goals and purpose can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).* — **Reward Hypothesis**: [2] p. 53

Articolare l'obiettivo dell'agente in termini di ricompensa espande significativamente la versatilità e l'applicabilità del MRP a una vasta gamma di problemi. La funzione di ricompensa agisce come un ponte tra gli obiettivi desiderati e le azioni che l'agente deve intraprendere per raggiungerli. Durante la fase di apprendimento, l'agente esplorerà e identificherà le strategie più efficaci per massimizzare la ricompensa accumulata, sviluppando così una comprensione più profonda dell'ambiente e delle azioni richieste per il successo.

Politecnico di Torino

REWARD IN EPISODIC VS CONTINUIG TASK

Nel campo del Reinforcement Learning, il concetto di *Expected Return* è centrale per l'obiettivo dell'agente, che consiste nel massimizzare la somma delle ricompense durante la durata di un episodio o di un'esperienza continua. La struttura della *reward function*, e di conseguenza il *return*, varia significativamente a seconda della natura specifica del task affrontato. In generale, i task nel RL possono essere categorizzati in due tipi principali:

1. **Episodic Task**: Questi task sono caratterizzati da una chiara definizione degli stati iniziali e terminali. Durante la fase di addestramento, l'interazione dell'agente con l'ambiente è suddivisa in distinte sequenze note come *episodi*.

   Ogni episodio rappresenta una traiettoria che inizia da uno stato iniziale e culmina in uno stato terminale. Al raggiungimento dello stato terminale, l'episodio si conclude, e il problema viene reinizializzato allo stato iniziale. In tali task, che includono giochi come gli scacchi, il *Expected Return* è definito come:

   $$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \ ... \ + R_T \qquad \text{Expected Return} \tag{4.2}$$

2. **Continuing Task**: Questi task sono caratterizzati dall'assenza di uno stato terminale definito, implicando un'interazione continua con l'ambiente. In questi casi, il concetto di episodio non si applica nel modo tradizionale.

   Il *return* per questi task può essere calcolato come la somma cumulativa delle ricompense. Tuttavia, dato che il compito è continuo, spesso si ricorre a un approccio scontato per evitare che il *return* diventi infinito. Ciò significa che le ricompense future sono scontate da un fattore che ne riduce la significatività col passare del tempo.

   $$G_t \doteq R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 R_{t+3} + \ ... \ = \sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1} \qquad \text{Discounted Return} \tag{4.3}$$

   Il parametro γ, noto come *discount rate*, è compreso tra $0 < \gamma < 1$. Il valore specifico di γ è determinato dalla natura del task[3] e rappresenta il grado di previsione desiderato nel modello.

   Un valore vicino allo zero renderebbe l'agente miope, concentrato solo sulle ricompense immediate, mentre un valore vicino a 1 lo renderebbe più attento alle ricompense future.

   Senza un *discount rate*, la somma delle ricompense future potrebbe non convergere, specialmente in ambienti con cicli infiniti. L'introduzione di un *discount rate* aiuta quindi a garantire la convergenza e la stabilità del modello.

Data la diversità intrinseca di questi tipi di task, può essere utile adottare una formulazione per il calcolo del *return* che accomuni sia i task episodici che quelli continui. Considerando la conclusione di un singolo episodio come l'entrata in uno specifico *absorbing state* che genera esclusivamente ricompense nulle, possiamo definire matematicamente il *Expected Return*, includendo anche la possibilità che $\gamma = 1$ e $T = \infty$, nel modo seguente:

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \qquad \text{Expected Return} \quad \begin{cases} \text{Episodic Task} \\ \text{Continuing Task} \end{cases} \tag{4.4}$$

[3] Generalmente il valore del *discount rate* è circa pari a: $\gamma \simeq 0.9$

Politecnico di Torino

STIMA DELLA VALUE FUNCTION

La stima delle *value function*, che indicano la bontà di uno stato (*state value*) o di un'azione (*action value*), è essenziale nei modelli di RL. Le principali differenze tra le varie classi di modelli di RL risiedono spesso nel modo in cui queste stime vengono effettuate e nelle tempistiche di aggiornamento della policy dell'agente.

Le *value function* sono espresse in funzione dell'*expected return*, poiché i reward che l'agente riceverà dipendono dagli stati in cui si trova lungo la traiettoria e dalle azioni che decide di intraprendere. Inoltre, le *value function* sono definite in relazione alla policy dell'agente, ovvero in funzione del particolare comportamento che l'agente adotta nell'interazione con l'ambiente. Vediamo quindi come lo stato, l'azione e la policy siano intrinsecamente interconnessi nel contesto del RL.

Definiamo quindi:

1. **State Value** - $v_\pi(s)$:

   La *value function* di uno stato s sotto la policy π è definita come l'*expected return* ottenuto dall'agente che, partendo dallo stato s, segue la policy π. Matematicamente, possiamo esprimerla nel modo seguente:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \quad \text{State Value} \tag{4.5}$$

2. **Action Value** - $q_\pi(s,a)$:

   Indica il *return*, ovvero la somma dei futuri *reward* che l'agente accumulerà partendo dallo stato s e compiendo un'azione a, successivamente seguendo le azioni dettate dalla policy π. Questo valore valuta l'efficacia di eseguire una specifica azione in uno stato dato, considerando le ricompense future ottenibili seguendo la policy, ed è espresso in funzione dell'*expected return*.

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s,\ A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s,\ A_t = a\right] \quad \text{Action Value} \tag{4.6}$$

Le *value function* presentano importanti proprietà e soddisfano relazioni ricorsive. La relazione tra gli *state value* di due stati consecutivi può essere espressa tramite l'equazione di Bellman per lo *state value*.

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma \cdot G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\Big[r + \gamma \cdot \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']\Big] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)\Big[r + \gamma \cdot v_\pi(s')\Big] \quad \forall s \in S \qquad \text{Bellman Equation for } v_\pi \end{aligned} \tag{4.7}$$

Risolvere un problema di RL significa identificare una policy π_* che guidi l'agente a massimizzare il reward accumulato lungo un episodio, ovvero trovare una policy ottimale.

Politecnico di Torino

Si definisce una policy π migliore o uguale a una policy π' se l'*expected return* associato a π è maggiore o uguale a quello associato a π' per ogni stato s. Questa relazione è fondamentale per comprendere come confrontare e valutare diverse policy:

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s) \qquad \forall s \in S$$

Tra tutte le policy possibili in uno specifico problema, esiste almeno una policy ottimale π_*, che massimizza l'*expected return* per ogni stato. Questa policy ottima condivide le stesse *value function* ottimali, che sono fondamentali per la sua identificazione e valutazione:

$$v_*(s) \doteq \max_\pi v_\pi(s) \qquad \text{Optimal State Value} \tag{4.8}$$

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a) \qquad \text{Optimal Action Value} \tag{4.9}$$

Queste *value function* ottimali sono fondamentali per la stima e l'aggiornamento delle policy. L'*Optimal State Value* fornisce una misura di quanto sia vantaggioso trovarsi in un particolare stato, considerando la migliore sequenza di azioni possibile da quel punto in poi. Analogamente, l'*Optimal Action Value* valuta l'efficacia di eseguire una specifica azione in uno stato dato, tenendo conto delle ricompense future derivanti dalla seguente migliore strategia di azione.

L'equazione di Bellman per le *value function* ottimali fornisce una relazione ricorsiva che collega il valore di uno stato all'*expected return* delle azioni possibili a partire da esso.

$$\begin{aligned}
v_*(s) &= \max_{a \in A(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s,\ A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma\, G_{t+1} \mid S_t = s,\ A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma\, v_*(S_{t+1}) \mid S_t = s,\ A_t = a] \\
&= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma \cdot v_*(s')] \qquad \text{Bellman Optimally Equation} - v_*
\end{aligned} \tag{4.10}$$

Considerando l'*action value* invece:

$$\begin{aligned}
q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \cdot \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s,\ A_t = a\Big] \\
&= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big] \qquad \text{Bellman Optimally Equation} - q_*
\end{aligned} \tag{4.11}$$

Per problemi caratterizzati da un numero di stati discreti, le equazioni di Bellman per le *value function* ottimali costituiscono un sistema di n equazioni in n incognite, dove n rappresenta il numero di stati nell'ambiente. Questo sistema descrive la relazione ricorsiva tra i valori degli stati e può essere risolto[4] per determinare la *value function* ottimale v_*, a condizione di conoscere con precisione le dinamiche dell'ambiente, ovvero le probabilità di transizione e le ricompense associate.

[4]Tramite metodi diretti, come l'eliminazione Gaussiana, oppure metodi numerici iterativi, come il metodo di Jacobi, Gauss-Seidel oppure il metodo del gradiente coniugato.

Politecnico di Torino

Una volta che la *state value function* ottimale v_* è stata determinata, è possibile ricavare anche la policy ottimale. Questo si fa esaminando, per ogni stato, quale azione massimizza l'*expected return* basato su v_*. In altre parole, per ogni stato s, si sceglie l'azione a che massimizza la somma della ricompensa immediata e del valore scontato dello stato successivo, come definito dall'equazione di Bellman ottimale.

Interessante notare che ogni policy che è *greedy* rispetto alla *value function* ottimale v_* è una policy ottimale. Una policy *greedy* in questo contesto sceglie, per ogni stato, l'azione che sembra migliore tra tutte quelle possibili basandosi sulla valutazione corrente della *value function*. Quindi, in pratica, determinare una policy *greedy* rispetto a v_* è spesso il passo finale dopo aver calcolato le *value function* ottimali.

I diagrammi di backup rappresentano visivamente il modo in cui le informazioni sui reward futuri e sulle *value function* vengono propagate all'indietro per aggiornare le stime correnti. Nel caso delle *value function* ottimali espresse dalle equazioni di Bellman:

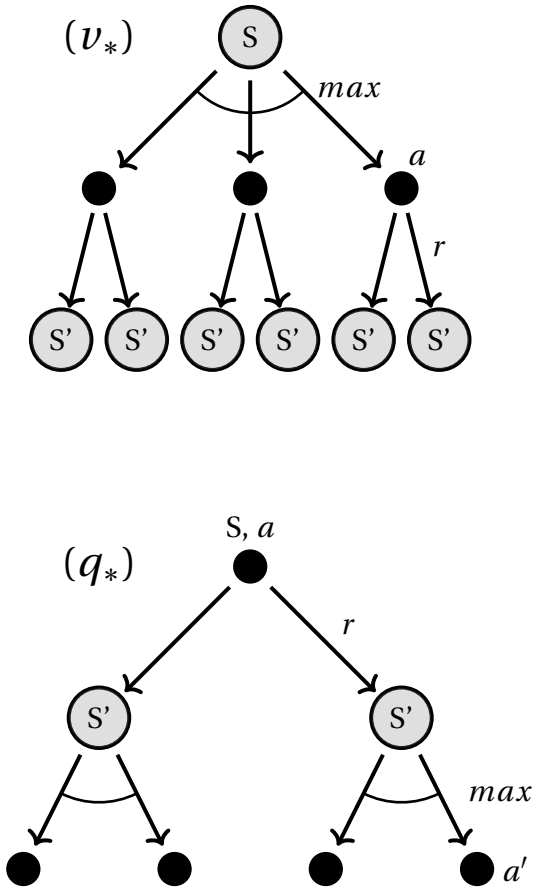


Figure 4.4: Backup Diagram per le value function ottimali - [2]

Questa relazione è al cuore del processo di ottimizzazione delle policy nel RL, poiché suggerisce che, per migliorare la stima del valore di uno stato, si deve considerare non solo il reward immediato delle azioni possibili, ma anche il valore dei futuri stati che queste azioni porteranno a raggiungere.

Politecnico di Torino

4.1.2 EXPLOITATION VS EXPLORATION

Per un determinato problema, il modello dovrà effettuare iterativamente una stima dell'*action value function*, in modo da poter successivamente arrivare a convergenza della *action value function* ottima. Definiamo quindi la stima del modello, ad un determinato *time step*, dell'*action value* come $Q_t(a)$.

$$Q_t(a) \doteq \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \qquad \text{Stima dell'Action Value} \tag{4.12}$$

Eseguendo queste stime, il modello identificherà ad ogni istante temporale e per ogni stato, almeno un'azione migliore[5] rispetto alle altre possibili da quello stato. Questa azione verrà identificata come *greedy action*, e l'agente, che sceglie di eseguire l'azione che ritiene più vantaggiosa, eseguirà un'*Exploitation* delle sue conoscenze.

Entriamo quindi in uno dei più discussi *trade-off* nei modelli di RL, la battaglia tra *Exploitation* ed *Exploration*.

1. **Exploitation**: Questa strategia implica che l'agente selezioni l'azione che, secondo il suo attuale stato di conoscenza (rappresentato da $Q_t(a)$), offre il maggiore *expected return*. L'*Exploitation* consente all'agente di massimizzare le ricompense basandosi sulle informazioni già acquisite, sfruttando le strategie note per ottenere il massimo beneficio immediato.

   Tuttavia, un approccio puramente basato sull'*Exploitation* può portare l'agente a rimanere bloccato in un massimo locale, soprattutto in ambienti complessi dove non tutte le azioni sono state esplorate e la stima attuale della *action value function* potrebbe non riflettere il vero potenziale di azioni non ancora provate.

2. **Exploration**: L'*Exploration*, d'altra parte, spinge l'agente ad esplorare azioni che non sono state selezionate frequentemente in passato o di cui si sa poco, nella speranza di scoprire nuove strategie che potrebbero portare a ricompense maggiori in futuro. Questo approccio è essenziale per evitare di rimanere intrappolati in strategie subottimali e per garantire che l'agente acquisisca una conoscenza più completa dell'ambiente.

   Tuttavia, l'*Exploration* comporta un costo, poiché scegliere un'azione meno conosciuta può portare a un risultato peggiore nel breve termine. Trovare il giusto equilibrio tra *Exploration* ed *Exploitation* è quindi fondamentale per il successo a lungo termine dell'agente in un ambiente di RL.

Il *trade-off* tra *Exploitation* ed *Exploration* è un dilemma centrale nel RL. Mentre l'*Exploitation* sfrutta le conoscenze correnti per ottenere ricompense immediate, l'*Exploration* cerca di migliorare la base di conoscenze dell'agente, potenzialmente portando a decisioni migliori e più informate nel futuro. La sfida consiste nel bilanciare questi due aspetti in modo dinamico, adattandosi alle esigenze e alle caratteristiche dell'ambiente in cui l'agente opera.

[5] Sempre in termini di *expected return*.

Politecnico di Torino

4.1.3 ACTION-VALUE METHODS

I metodi più semplici per la scelta delle azioni nel Reinforcement Learning, noti come *action-value methods*, si basano sulle stime delle action value. Ricordiamo che il valore di un'azione è il ritorno medio che si prevede di ricevere quando quell'azione viene scelta.

1. **Sample-Average Method**:

   Questo metodo stima il valore di un'azione come la media dei reward precedentemente ricevuti quando quella stessa azione è stata selezionata. Matematicamente, se un'azione a è stata scelta n volte fino al tempo t, e ha restituito i reward $R_1, R_2, \ldots, R_n$, il suo valore è stimato come:

   $$Q_t(a) = \frac{1}{n} \sum_{i=1}^{n} R_i$$

   Questo metodo è semplice e intuitivo, ma richiede una storia di tutte le ricompense passate per ogni azione, il che può diventare poco pratico in ambienti con molti step o molte azioni. Inoltre, è adatto a problemi stazionari dove l'ambiente è deterministico e le transizioni non sono casuali. L'implementazione del sample-average method può essere rimodellata come segue per ridurre la memoria richiesta:

   $$Q_{n+1} = Q_n + \frac{1}{n}\big[R_n - Q_n\big] \qquad \text{Sample-Average Method} \tag{4.13}$$

   Questa espressione può essere interpretata come un aggiornamento della stima precedente basato su uno specifico *step-size*:

   $$\text{New\_Estimate} \longleftarrow \text{Old\_Estimate} + \text{Step\_Size}\big[\text{Target} - \text{Old\_Estimate}\big]$$

   Per problemi non stazionari, dove la distribuzione dei reward cambia nel tempo, il modello deve dare un peso minore alle stime precedenti. Questo si ottiene utilizzando un parametro di step-size costante $\alpha \in (0,1]$:

   $$Q_{n+1} = Q_n + \alpha\big[R_n - Q_n\big] \qquad \text{Exponential Recency-Weighted Average} \tag{4.14}$$

   La *Stochastic Approximation Theory* fornisce le condizioni necessarie per garantire la convergenza assoluta, assicurando che lo step-size sia sufficientemente ampio per superare le fluttuazioni casuali iniziali e abbastanza piccolo per garantire la convergenza del metodo:

   $$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \qquad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$$

2. ε**-greedy Policy**:

   La politica ε-greedy è un approccio per bilanciare exploitation ed exploration. Con probabilità $1-\varepsilon$ (exploitation), l'agente sceglie l'azione con il più alto valore stimato (azione greedy). Con probabilità ε (exploration), l'agente sceglie un'azione a caso tra tutte quelle disponibili. Il valore di ε determina il grado di exploration: un ε alto promuove l'esplorazione, mentre un ε basso favorisce l'exploitation. Questa politica permette all'agente di continuare ad esplorare l'ambiente mentre sfrutta progressivamente ciò che ha imparato.

Politecnico di Torino

UPPER CONFIDENCE BOUND - UCB

Il metodo Upper Confidence Bound (UCB) è un'altra strategia per affrontare il trade-off tra exploitation ed exploration, che si differenzia dalla politica ε-greedy per il suo approccio più informato e meno casuale.

Invece di esplorare casualmente, UCB modifica la stima dell'action value basandosi non solo sulla ricompensa media, ma anche sulla frequenza con cui ogni azione è stata selezionata. Matematicamente, UCB seleziona l'azione a al tempo t massimizzando il seguente criterio:

$$A_t \doteq \underset{a}{arg\,max}\Big[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}\Big] \qquad \text{UCB} \tag{4.15}$$

Dove $N_t(a)$ è il numero di volte che l'azione a è stata scelta fino al tempo t, e c è un coefficiente che determina il grado di exploration. Il termine addizionale $c\sqrt{\frac{\ln t}{N_t(a)}}$ agisce come un bonus di confidenza che aumenta il valore stimato per le azioni meno sperimentate, spingendo l'agente ad esplorarle più frequentemente. Con il tempo, man mano che l'agente apprende di più sull'ambiente, questo termine diventa meno influente, e l'agente si affida maggiormente alle sue stime dirette dei valori delle azioni.

GRADIENT BANDIT ALGORITHMS

I Gradient Bandit Algorithms rappresentano una categoria di metodi per la scelta delle azioni che si basano sui principi del gradiente per ottimizzare la policy. A differenza dei metodi ε-greedy o UCB, che selezionano le azioni in base a stime di valore, gli algoritmi di tipo Gradient Bandit assegnano preferenze a ciascuna azione e utilizzano queste preferenze per determinare la probabilità di selezionare ciascuna azione. L'idea fondamentale è quella di aumentare la preferenza per le azioni che hanno portato a reward migliori di quanto atteso, e diminuire la preferenza per quelle che hanno deluso.

Formalmente, se $H_t(a) \in \mathbb{R}$ rappresenta la preferenza per l'azione a al tempo t, la probabilità che l'azione a sia selezionata è data da una distribuzione di probabilità softmax[6] sulle preferenze:

$$Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a) \qquad \text{Gradient Bandit} \tag{4.16}$$

Dove k è il numero totale di azioni. L'agente aggiorna le preferenze $H_t(a)$ ogni volta che sceglie un'azione e riceve un reward.

Il metodo più comune per aggiornare le preferenze è il *policy gradient algorithm*. Questo algoritmo aggiorna le preferenze in direzione che aumenta il valore atteso del reward. L'aggiornamento si basa sulla differenza tra il reward ricevuto e una stima del reward atteso, chiamato *baseline*, per ridurre la varianza degli aggiornamenti. La preferenza per l'azione selezionata viene incrementata in proporzione a questa differenza, mentre le preferenze per tutte le altre azioni vengono decrementate. Matematicamente, l'aggiornamento delle preferenze per l'azione a scelta al tempo t:

$$H_{t+1}(a) = H_t(a) + \alpha(R_t - \bar{R}_t)(1 - \Pr(A_t = a))$$

$$H_{t+1}(b) = H_t(b) - \alpha(R_t - \bar{R}_t)\Pr(A_t = b)$$

[6] La distribuzione di probabilità softmax è una generalizzazione della funzione logistica e serve a trasformare un vettore di valori reali in un vettore di valori che rappresentano probabilità, in modo che la somma di queste probabilità sia pari a 1.

Politecnico di Torino

Per tutte le altre azioni $b \neq a$. Qui, α è un parametro di step-size, R_t è il reward ricevuto, e $\bar{R}_t$ è il baseline.

I Gradient Bandit Algorithms sono particolarmente utili in ambienti con un grande numero di azioni, dove la stima diretta del valore di ciascuna azione potrebbe essere impraticabile. Inoltre, la capacità di apprendere preferenze relative piuttosto che valori assoluti consente a questi algoritmi di adattarsi efficacemente anche in ambienti non stazionari. Tuttavia, la scelta dei parametri come α e il baseline $\bar{R}_t$ può influenzare significativamente le prestazioni e richiede attenzione durante la configurazione dell'algoritmo.

Utilizzare quindi le equazioni di Bellman per risolvere problemi di Reinforcement Learning (RL) equivale, in un certo senso, a condurre un'analisi esaustiva dello spazio delle possibilità. Tale approccio calcola le probabilità e valuta ogni possibile esito in termini di *expected return*. Tuttavia, questo metodo presuppone il soddisfacimento di tre importanti condizioni che, purtroppo, risultano spesso irrealizzabili nei contesti pratici:

1. **Conoscenza delle Dinamiche dell'Ambiente**: Per applicare efficacemente le equazioni di Bellman, è fondamentale avere una comprensione completa delle reazioni dell'ambiente alle azioni dell'agente, incluso il dettaglio delle probabilità di transizione e delle ricompense associate. Nella realtà, tale livello di conoscenza è raramente disponibile, specialmente in ambienti complessi o imprevedibili.

2. **Risorse Computazionali Adeguati**: L'attuazione delle equazioni di Bellman richiede un'intensa capacità computazionale, particolarmente in ambienti con un vasto numero di stati o azioni. La necessità di analizzare ogni possibile stato e azione rende il processo insostenibile all'aumentare della complessità del problema.

3. **Proprietà di Markov degli Stati**: Le equazioni di Bellman presuppongono che gli stati abbiano la proprietà di Markov, dove il futuro è completamente determinato dallo stato presente indipendentemente dal passato. Questa assunzione è spesso inapplicabile in ambienti reali, dove la storia e la complessità non sono interamente catturate dallo stato corrente.

Frequentemente, è la necessità di risorse computazionali estensive a rappresentare il primo ostacolo. Si tenta spesso di affrontare problemi su larga scala con uno spazio degli stati talmente ampio[7] da rendere impraticabile l'esecuzione di tutti i calcoli necessari tramite i metodi tradizionali basati sulle equazioni di Bellman.

Questi vincoli hanno stimolato la ricerca e lo sviluppo di metodologie alternative, più efficienti e praticabili. Algoritmi come il Q-learning e metodi che si avvalgono di funzioni di approssimazione, quali le reti neurali (NN), rappresentano come il reinforcement learning possa essere adattato a contesti più ampi e complessi senza aderire rigidamente alle assunzioni classiche delle equazioni di Bellman.

[7] Consideriamo, per esempio, il gioco del backgammon con uno spazio degli stati dell'ordine di 10^{20}.

Politecnico di Torino

4.2 Tabular Solution Methods

I metodi di soluzione tabulari sono fondamentali per la loro chiarezza concettuale e l'applicabilità a problemi con spazi di stato e azione discreti e limitati. Questi metodi, che utilizzano tabelle per rappresentare e aggiornare i valori associati agli stati e alle azioni, non solo hanno stabilito le basi per tecniche più avanzate, ma sono stati anche tra i primi sviluppi nel RL. Hanno quindi aperto la strada allo sviluppo di metodi decisionali più complessi, servendo come fondamento per tecniche successive più sofisticate.

Questi metodi, includono:

1. **Dynamic Programming (DP)**: Una suite di algoritmi che risolvono problemi di decisione sequenziale sfruttando la programmazione dinamica. DP è particolarmente potente quando il modello dell'ambiente è completamente noto, ma soffre quando la dimensione dello spazio degli stati diventa grande.
2. **Monte Carlo (MC) Methods**: Questi metodi stimano i valori direttamente dall'esperienza, campionando sequenze complete di stati e ricompense. Non richiedono la conoscenza del modello dell'ambiente e possono essere applicati a problemi con spazi degli stati più ampi rispetto a DP.
3. **Temporal-Difference (TD) Learning**: Una classe di metodi che combinano le idee di DP e MC. TD Learning aggiorna le stime dei valori parzialmente sulla base di stime esistenti, senza attendere l'esito finale di un episodio. Questo rende TD Learning più efficiente e adatto a problemi in tempo reale e non episodici.

La caratteristica distintiva dei metodi tabulari è la loro capacità di fornire una soluzione esatta per problemi di RL finiti alla Markov, dove ogni decisione è presa in uno spazio discreto e ogni stato ha un valore ben definito che può essere appreso e memorizzato. Mentre in ambienti più complessi e di alta dimensione si rendono necessari metodi di approssimazione, nei contesti in cui l'uso di tabelle è praticabile, i metodi tabulari possono apprendere il vero valore di ogni stato o coppia stato-azione, conducendo a politiche ottimali attraverso un processo iterativo di aggiornamento.

4.2.1 Generalized Policy Iteration - GPI

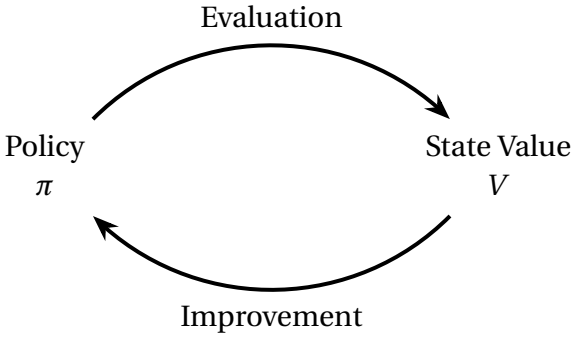
Generalized Policy Iteration (GPI) rappresenta il cuore concettuale dietro numerosi metodi tabulari. GPI descrive un quadro in cui la valutazione della policy e il suo miglioramento avvengono in un ciclo continuo, alimentandosi reciprocamente fino alla convergenza verso una policy ottimale.

GPI non è legato a un singolo algoritmo specifico. Piuttosto, rappresenta una strategia generale che può essere adottata e adattata da diversi metodi tabulari nel RL. Questa flessibilità rende GPI un principio fondamentale che attraversa vari approcci e algoritmi nel campo, da quelli più semplici e diretti a quelli più complessi e sofisticati.

Politecnico di Torino

POLICY EVALUATION AND IMPROVEMENT IN GPI

Il cuore di GPI è l'iterazione tra due processi fondamentali: la policy evaluation e la policy improvement.



1. *Policy Evaluation*: In questa fase, il valore di una policy data viene stimato. La policy evaluation può essere realizzata attraverso diversi metodi, a seconda del contesto e dell'algoritmo specifico in uso. Il risultato è una comprensione più profonda dell'efficacia della policy attuale.
2. *Policy Improvement*: Sulla base della valutazione effettuata, si cerca di produrre una nuova policy che sia in qualche modo migliore della precedente. Questo miglioramento può essere attuato in modi diversi, ma l'obiettivo comune è di produrre una policy che massimizzi meglio i reward attesi.

Questi due processi si alternano in un ciclo continuo, con la policy che viene progressivamente affinata ad ogni iterazione. L'approccio iterativo di GPI assicura che, anche se ogni singolo passo potrebbe non produrre la policy ottimale, la policy continuerà a migliorare fino a raggiungere l'ottimalità.

APPLICAZIONE DI GPI NEI METODI TABULARI

Metodi come Dynamic Programming (DP), Monte Carlo (MC), e Temporal-Difference (TD) Learning implementano tutti il framework di GPI, ma con variazioni e specificità che li rendono adatti a differenti tipi di problemi e contesti.

- **Dynamic Programming**: Il DP implementa la GPI attraverso un approccio strutturato che richiede una conoscenza dettagliata delle dinamiche dell'ambiente. La GPI nel DP alterna tra valutazione rigorosa e miglioramento sistematico delle policy, adattandosi a spazi di stato e azione discreti e ben definiti
- **Monte Carlo Methods**: I metodi MC applicano la GPI in modo che sfrutta esperienze dirette e complete, senza la necessità di conoscere in anticipo le dinamiche dell'ambiente. La GPI nei metodi MC è centrata sull'analisi di interi episodi per valutare e migliorare le policy in maniera empirica.
- **Temporal-Difference Learning**: Il TD Learning fonde concetti da DP e MC, aggiornando le stime dei valori basandosi su esperienze parziali e stime esistenti. La GPI nel TD Learning consente un apprendimento flessibile e in tempo reale, efficace in ambienti dinamici e non stazionari.

In sintesi, la GPI fornisce un quadro per la valutazione continua e il miglioramento delle policy, essendo il motore centrale dietro molti approcci al Reinforcement Learning.

Politecnico di Torino

4.2.2 DYNAMIC PROGRAMMING - DP

La Programmazione Dinamica (DP) risulta particolarmente efficace nell'affrontare Processi Decisionali di Markov (MDP) con spazi di stato e azione discreti e ben definiti. Questa tecnica decompositiva scompone problemi complessi in sotto-problemi più gestibili, risolvendoli attraverso un approccio ricorsivo.

Gli algoritmi che appartengono a questa classe, dato un problema, cercano di ottenere la policy ottima sfruttando la conoscenza dell'ambiente. Questa caratteristica comporta il più grande limite di questa categoria, infatti per problemi di interesse pratico risulta difficile creare un perfetto modello dell'ambiente e genericamente sfocia in un grosso costo computazionale.

L'idea chiave è quella di sfruttare le equazioni di Bellman per la state value function ottima e l'action value function ottima per cercare di ottenere la policy ottimale al problema. Il modo di operare della DP, che sembrerebbe adattarsi bene solo a problemi caratterizzati da spazi decisionali discreti, può essere applicata anche a spazi decisionali continui, trovando soluzioni approssimate, discretizzando lo spazio degli stati e delle azioni.

DYNAMIC PROGRAMMING PREDICTION

La valutazione della policy, nel contesto del Dynamic Programming (DP), è un processo necessario per determinare l'efficacia di una strategia specifica in termini di ritorno atteso. Questo processo si avvale dell'Iterazione di Bellman, una relazione ricorsiva che collega il valore di uno stato al valore dei suoi stati successivi. L'Iterazione di Bellman rappresenta una delle pietre miliari della Programmazione Dinamica e del Reinforcement Learning in generale, fornendo un meccanismo formale per valutare e migliorare le policy.

La valutazione della policy π per uno stato s può essere formalmente espressa attraverso l'equazione di Bellman come segue:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi\Big[G_t \mid S_t = s\Big] \\ &= \mathbb{E}_\pi\Big[R_{t+1} + \gamma \cdot G_{t+1} \mid S_t = s\Big] \\ &= \mathbb{E}_\pi\Big[R_{t+1} + \gamma \cdot v_\pi(S_t + 1) \mid S_t = s\Big] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \qquad \text{Policy Evaluation} \end{aligned} \tag{4.17}$$

Dove $v_\pi(s)$ rappresenta il valore dello stato s sotto la policy π, γ è il fattore di sconto che modula l'importanza dei ritorni futuri, e $p(s', r|s, a)$ descrive la probabilità di transizione dallo stato s allo stato s' con ricompensa r dopo aver preso l'azione a.

Politecnico di Torino

In contesti in cui la dinamica dell'ambiente è completamente nota, la Policy Evaluation si traduce nella risoluzione di un sistema di n equazioni lineari in n incognite, dove n rappresenta il numero di stati. L'approccio iterativo[8], conosciuto come *Iterative Policy Evaluation*, parte da un'approssimazione iniziale arbitraria v_0 e procede fino alla convergenza, dove $v_k \to v_\pi$ quando $k \to \infty$, come mostrato nella seguente equazione:

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi\Big[R_{t+1} + \gamma \cdot v_k(S_{t+1}) | S_t = s\Big] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma \cdot v_k(s')\Big] \qquad \text{Bellman Iteration} \end{aligned} \tag{4.18}$$

Il termine $v_k(s')$ rappresenta l'approssimazione al passo k del valore dello stato s'. L'equazione 4.21 illustra come ogni stima v_{k+1} sia calcolata basandosi sull'approssimazione precedente v_k, implementando ciò che è noto come *expected update*.

È importante sottolineare che l'efficienza computazionale della Policy Evaluation dipende dalla dimensione dello spazio degli stati e dalla complessità delle dinamiche dell'ambiente. Inoltre, varianti come la *value iteration* e la *policy iteration* migliorano l'efficacia di questo processo in scenari più complessi.

DYNAMIC PROGRAMMING ESTIMATION

Dopo aver valutato una policy, il passo successivo è migliorarla. Questo processo, noto come *Policy Improvement*, coinvolge l'aggiornamento delle scelte dell'agente per massimizzare il ritorno atteso. Infatti vorremmo valutare, una volta calcolata v_π, se possa essere conveniente in determinati stati s, eseguire un'azione $a \neq \pi(s)$, ovvero un'azione differente rispetto all'azione scelta dalla policy π.

Il *Policy Improvement Theorem* gioca un ruolo vitale in questo contesto. Esso fornisce una base formale per verificare se, in determinati stati s, eseguire un'azione $a \neq \pi(s)$ può effettivamente portare a un miglioramento. L'azione a viene confrontata con l'azione suggerita dalla policy corrente π attraverso un calcolo che coinvolge l'action value function $q_\pi(s, a)$ e la state value function $v_\pi(s)$. Il teorema può essere enunciato come segue:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \quad \Longrightarrow \quad v_{\pi'}(s) \geq v_\pi(s) \qquad \text{Policy Improvement Theorem} \tag{4.19}$$

L'Equazione afferma che, se esiste un'azione che ha un valore atteso maggiore del valore dello stato attuale sotto la policy π, allora adottare questa nuova azione nella policy migliorata π' porterà a un rendimento maggiore o uguale rispetto alla policy corrente. In altre parole, si garantisce che non si peggiora mai selezionando azioni che massimizzano l'action value.

[8] I metodi iterativi sono:

1. Jacobi: Questo metodo utilizza due array per salvare il vecchio ed il nuovo valore della state value function.
2. Gauss-Seidel: Questo metodo utilizza un solo array per salvare il nuovo valore della state value function, eseguendo gli updates "in place".
3. Metodo del Gradiente Coniugato Precondizionato - PCG

Politecnico di Torino

Applicando questo principio a tutti gli stati, possiamo formulare un algoritmo iterativo per il miglioramento della policy. Ad ogni iterazione, la policy viene aggiornata scegliendo, per ogni stato, l'azione che offre il massimo valore secondo l'action value function. Matematicamente, il processo di miglioramento della policy può essere descritto come:

$$\pi'(s) \doteq \arg\max_a q_\pi(s,a) \qquad \text{Policy Improvement Step} \tag{4.20}$$

Dove π' rappresenta la nuova policy, presumibilmente migliorata rispetto alla precedente. L'Equazione illustra come, in ogni stato, si selezioni l'azione che si prevede porti al maggior ritorno. Con un'applicazione ripetuta di valutazione e miglioramento, la policy evolve, convergendo progressivamente verso una strategia ottimale.

È importante sottolineare che la Policy Improvement è un processo iterativo e incrementale. Ogni ciclo di valutazione e miglioramento non garantisce immediatamente l'ottimalità, ma assicura un passo verso una policy che massimizza i ritorni. Inoltre, sebbene il framework teorico sia solido, l'implementazione pratica di questi principi può incontrare sfide, soprattutto in ambienti con spazi di stato e azione di grandi dimensioni. Strategie come la *Value Iteration* e la *Policy Iteration* sono esempi di come la DP affronta questi problemi, semplificando e accelerando il processo di convergenza.

POLICY AND VALUE ITERATION

Negli algoritmi della DP, due metodi fondamentali emergono per trovare la policy ottimale in un Processo Decisionale di Markov (MDP): la Policy Iteration e la Value Iteration.

La Policy Iteration è un approccio iterativo che alterna tra valutazione e miglioramento della policy, convergendo verso una policy che massimizza i ritorni attesi. Iniziando da una policy arbitraria, il processo continua ad aggiornare la policy finché non si verificano più cambiamenti significativi, indicando la convergenza.

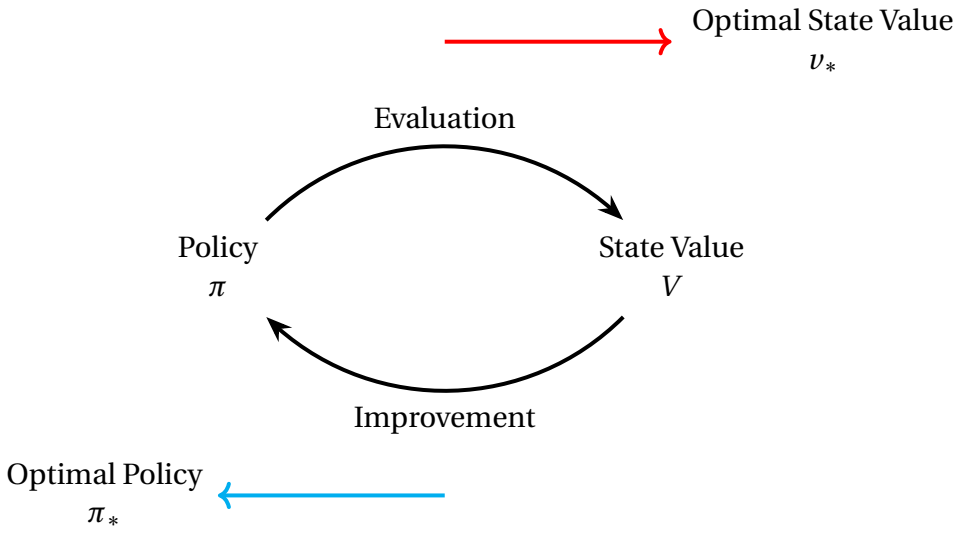


Figure 4.5: DP GPI (*Generalized Policy Iteration*)

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_*}$$

Politecnico di Torino

Il processo segue questi passaggi:

1. Inizia con una policy arbitraria π_0.
2. **Policy Evaluation**: Valuta la policy corrente π_i calcolando il suo stato di value function $v_{\pi_i}(s)$ per ogni stato s attraverso l'iterazione di Bellman.
3. **Policy Improvement**: Migliora la policy attuale π_i scegliendo, per ogni stato, l'azione che massimizza l'action value function calcolata nella valutazione.
4. Ripete i passaggi di valutazione e miglioramento fino alla convergenza a una policy ottimale π_*.

Nei MDP finiti, dove esistono un numero finito di policy, la policy iteration converge alla policy ottima con un numero finito di iterazioni.

Lo svantaggio della di questa continua iterazione è che nel momento in cui dobbiamo valutare la policy (*prediction*) l'algoritmo dovrà effettuare calcoli iterativi che richiedono un certo tempo per essere svolti ed un carico computazionale oneroso. Per aggirare questo inconveniente potremmo arrestare la convergenza impostando una tolleranza maggiore oppure in base a constraint temporali, senza però perdere la caratteristica di convergenza del metodo.

Il caso limite di questo concetto sarebbe quello di stoppare la policy evaluation dopo un singolo step iterativo, questo particolare algoritmo viene chiamato *value iteration*.

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}_\pi \Big[R_{t+1} + \gamma \cdot v_k(S_{t+1}) | S_t = s \Big] \\ &= \max_a \sum_{s',r} p(s', r|s, a) \Big[r + \gamma \cdot v_k(s') \Big] \qquad \text{Value Iteration} \end{aligned} \tag{4.21}$$

ASYNCHRONOUS DYNAMIC PROGRAMMING

L'Asynchronous Dynamic Programming (ADP) rappresenta un'avanzata metodologia nel campo della Programmazione Dinamica, concepita per mitigare i vincoli computazionali nei contesti di ampia scala. Divergendo dai metodi convenzionali che richiedono aggiornamenti simultanei per ogni stato in ciascuna iterazione, l'ADP adotta una strategia selettiva e flessibile, focalizzandosi sull'aggiornamento dei valori di un sottoinsieme qualsiasi di stati per volta, trascurando talvolta stati che non influenzano significativamente la soluzione del problema.

Un vantaggio distintivo dell'ADP è la sua capacità di garantire la convergenza alla value function ottimale v_*, a condizione che il fattore di sconto γ sia compreso tra 0 e 1 ([2]). Questa proprietà assicura che, nonostante l'aggiornamento asincrono e selettivo, l'algoritmo si avvicinerà progressivamente alla soluzione ottimale.

Questa modalità di operare si rivela particolarmente utile in ambienti con spazi degli stati vasti, dove l'esecuzione di aggiornamenti completi risulta inattuabile. L'ADP consente agli algoritmi di adattarsi prontamente alle nuove informazioni senza la necessità di ricalcolare l'intera policy. Tuttavia, per assicurare un apprendimento bilanciato e comprensivo, è cruciale una selezione accurata degli stati da aggiornare. Le strategie per determinare quali stati aggiornare possono variare considerevolmente, estendendosi da tecniche randomizzate a metodi più raffinati basati sull'esperienza accumulata o sull'analisi dettagliata della struttura intrinseca del problema.

Politecnico di Torino

EFFICIENZA DELLA DP

Nonostante la sua potenza e versatilità, la DP è soggetta a significative sfide di efficienza, particolarmente in ambienti con un ampio spettro di stati o azioni. Questo fenomeno, noto come *curse of dimensionality*, descrive l'aumento esponenziale della complessità computazionale con l'aggiunta di ogni nuova dimensione allo spazio degli stati o delle azioni, rendendo la DP impraticabile per molti problemi realistici nel RL.

La necessità di una conoscenza perfetta dell'ambiente è spesso irrealizzabile in applicazioni pratiche, dove l'incertezza e l'imprevedibilità giocano ruoli significativi. Per superare queste sfide, sono state sviluppate diverse strategie. Tra queste, l'uso di tecniche di approssimazione, dove le funzioni di valore sono approssimate piuttosto che calcolate esattamente, e metodi di riduzione della dimensionalità, che cercano di semplificare il problema mantenendo informazioni critiche.

Altri approcci includono l'integrazione della DP con metodi di Deep Learning, capaci di catturare strutture complesse e approssimare le funzioni di valore con efficienza notevolmente migliorata. Inoltre, l'adozione di strategie asincrone nella DP può contribuire a mitigare i problemi computazionali, consentendo aggiornamenti più mirati e frequenti che accelerano la convergenza verso la policy ottimale.

Dynamic Programming Algorithm	
DP	- Spazi decisionali: continui e discreti - Modello dell'ambiente completo richiesto - Aggiornamenti basati sul valore atteso (*expected updates*) - Implementazione di GPI con equazioni di Bellman - Sweep uniforme durante l'aggiornamento - Applicabile principalmente a problemi di piccole dimensioni
ADP	- Spazi decisionali: continui e discreti - Modello dell'ambiente: può essere meno dettagliato - Aggiornamenti asincroni senza attesa per il completamento di tutti gli stati - Utilizzo delle equazioni di Bellman in modo asincrono - Sweep asincrono: aggiornamento basato su criteri specifici anziché uniforme - Maggiore flessibilità e potenzialmente più efficace su problemi di dimensioni maggiori

Table 4.2: Confronto tra Programmazione Dinamica e Programmazione Dinamica Asincrona

Politecnico di Torino

4.2.3 MONTE CARLO METHODS - MC

I Metodi Monte Carlo (MC) nel Reinforcement Learning costituiscono una categoria fondamentale di algoritmi distintiva per la loro capacità di apprendere direttamente dall'esperienza, senza richiedere una comprensione dettagliata del modello dell'ambiente. Questi metodi si avvalgono del campionamento casuale per stimare soluzioni a problemi decisionali, proponendo un'alternativa agli algoritmi basati sulla Programmazione Dinamica (DP).

Contrariamente ai metodi DP, che presuppongono un modello esatto delle dinamiche ambientali, i metodi MC consentono all'agente di imparare interagendo direttamente con l'ambiente. Tale interazione permette all'agente di trarre insegnamenti dai propri errori[9] e dalle esperienze accumulate. I metodi MC valutano le *value functions* al termine di interi episodi anziché step-by-step, consentendo all'agente di aggiornare la propria conoscenza delle funzioni di valore e delle policy in base alla media delle ricompense cumulative ricevute.

Questi metodi sono quindi applicabili in scenari dove una conoscenza completa dell'ambiente non è disponibile o in cui la complessità del modello supera i limiti pratici di computazione. Imparando direttamente dall'esperienza attraverso il campionamento di traiettorie complete di stati, azioni e ricompense, i metodi MC sono efficaci in un'ampia gamma di problemi di RL. Questa versatilità li rende strumenti potenti, capaci di fornire stime accurate delle value function anche in ambienti complessi e non deterministici (non-stazionari).

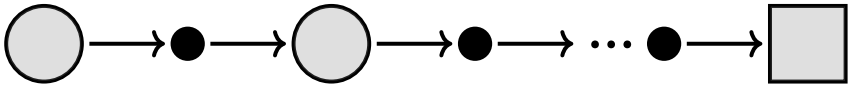


Figure 4.6: MC backup diagram

MONTE CARLO PREDICTION O ESTIMATION

La *Monte Carlo Prediction*, o valutazione della policy, nei metodi Monte Carlo, si concentra sulla stima delle value function associate a una policy specifica attraverso il campionamento di episodi. A differenza dei metodi DP, i metodi MC non richiedono la conoscenza preventiva delle dinamiche dell'ambiente. Invece, questi metodi calcolano la media dei ritorni per ogni stato visitato per approssimare il suo valore atteso.

Un tratto distintivo della *Monte Carlo prediction* è che la valutazione delle value functions di uno stato avviene solamente al termine di ciascun episodio, rendendo questi metodi particolarmente adatti per ambienti episodici. I metodi MC si differenziano in base al trattamento delle visite multiple allo stesso stato all'interno di un singolo episodio:

1. **First-Visit MC**: Questo algoritmo considera il return associato solo alla prima visita di uno stato durante un episodio. Ogni return è indipendente e fornisce una stima identicamente distribuita della value function $v_\pi(s)$, con varianza finita.

2. **Every-Visit MC**: A differenza del First-Visit MC, questo algoritmo considera tutte le visite a uno stato durante un episodio. Calcola inizialmente una media di tutti i returns associati a un determinato stato in un singolo episodio e poi calcola una media di queste medie su tutti gli episodi. Sebbene più complesso del First-Visit MC, la convergenza dell'Every-Visit MC è più rapida.

[9] Come, ad esempio, grazie a delle ricompense negative.

Politecnico di Torino

Entrambe le varianti convergono alla value function $v_\pi(s)$ sotto l'ipotesi di un numero infinito di visite allo stato. La valutazione della policy tramite metodi MC è espressa dalla seguente formula:

$$q_\pi(s,a) = \frac{\sum_{t \in T(s)} G_t}{|T(s)|} \qquad \text{Monte Carlo Prediction} \tag{4.22}$$

Dove $T(s)$ denota l'insieme dei tempi in cui lo stato s è stato visitato, $|\dot{|}$ indica la cardinalità e G_t rappresenta il ritorno totale a partire dal tempo t.

Da sottolineare è il fatto che, contrariamente ai metodi DP e TD, i metodi MC non utilizzano stime precedenti per aggiornare le funzioni di valore (*no Bootstrap*), rendendo ogni episodio indipendente dagli altri. Questa caratteristica distingue nettamente i metodi MC dai metodi TD. Inoltre, il costo computazionale per la stima delle funzioni di valore nei metodi MC è indipendente dalla grandezza dello spazio degli stati, fornendo flessibilità e praticità di applicazione.

Nell'ambito dei metodi MC, la stima dell'action value è particolarmente rilevante poiché fornisce una valutazione diretta dell'efficacia delle azioni in contesti specifici, basata sui risultati ottenuti dall'agente attraverso l'esplorazione diretta dell'ambiente. Questa stima si ottiene analizzando i returns ricevuti in seguito alle azioni intraprese, consentendo all'agente di identificare e privilegiare le azioni che hanno portato a ricompense maggiori.

Il processo di stima nell'ambito dei metodi MC segue una struttura ben definita:

1. L'agente esplora l'ambiente seguendo una policy, raccogliendo dati sulle sequenze di stati, azioni e ricompense.
2. Per ciascuna azione intrapresa, si calcola il ritorno totale dall'inizio dell'azione fino al termine dell'episodio.
3. I ritorni per ciascuna azione vengono mediati per fornire una stima dell'action value function $q_\pi(s,a)$.

Questo approccio permette di creare una mappa empirica del valore di ciascuna azione in vari contesti, fornendo un modo efficace per sviluppare strategie decisionali in scenari incerti e complessi. Tuttavia, una sfida significativa con i metodi MC è la potenziale mancanza di osservazioni per tutte le coppie stato-azione, specialmente in policy deterministiche dove l'agente potrebbe ripetutamente scegliere la stessa azione in uno stato specifico, trascurando altre azioni potenzialmente vantaggiose.

Per affrontare il dilemma dell'*exploration vs exploitation*, fondamentale nel RL, è essenziale garantire un'adeguata esplorazione. Una strategia per assicurare che ogni coppia stato-azione venga esplorata è l'uso di *exploring starts*, che introduce una probabilità non nulla di selezionare qualsiasi azione in uno stato iniziale, favorendo così l'esplorazione.

In policy deterministiche, dove la sfida dell'esplorazione continua è più evidente, l'uso di *exploring starts* è cruciale. Al contrario, in policy stocastiche, dove le azioni sono scelte secondo una distribuzione di probabilità, il problema dell'esplorazione è intrinsecamente mitigato. La flessibilità dei metodi MC nel gestire sia policy deterministiche che stocastiche li rende strumenti efficaci per affrontare problemi decisionali complessi in ambienti variabili e incerti.

Politecnico di Torino

MONTE CARLO CONTROL

Il *Control* rappresenta l'approccio sfruttato dai metodi MC per ottimizzare le policy tramite l'impiego della Generalized Policy Iteration (GPI). Questo metodo utilizza esperienze campionate in episodi completi per aggiornare e perfezionare iterativamente le decisioni prese dall'agente. Il processo di controllo si articola nei seguenti passaggi chiave:

1. **Valutazione della Policy**: In linea con i principi dei metodi Monte Carlo Prediction, l'agente valuta la policy attuale analizzando dati da episodi completi. Ciò include il calcolo delle stime dell'action value function per ogni coppia stato-azione osservata.

$$q_\pi(s,a) = \frac{\sum_{t \in T(s)} G_t}{|T(s)|}$$

2. **Miglioramento della Policy**: Sulla base delle stime ottenute, l'agente rivede la sua policy, preferendo le azioni che massimizzano l'action value. Questo aggiornamento può avvenire attraverso un approccio greedy o strategie più complesse che equilibrano exploration ed exploitation.

$$\pi(s) \doteq arg\ \max_a q(s,a)$$

3. **Iterazione del Processo**: Valutazione e miglioramento si susseguono iterativamente. Ad ogni ciclo, la policy si aggiorna con le nuove esperienze, conducendo gradualmente verso una policy più efficace.

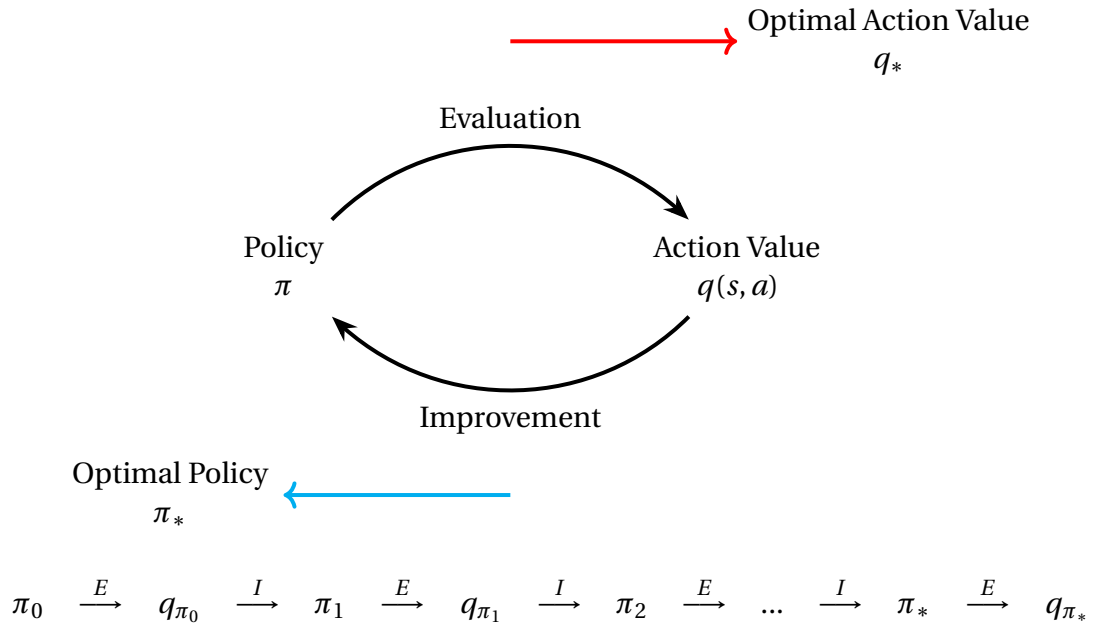


Figure 4.7: Monte Carlo GPI (*Generalized Policy Iteration*)

Politecnico di Torino

Sebbene la convergenza dei metodi Monte Carlo sia teoricamente garantita in determinate condizioni, l'applicazione pratica di questi metodi presenta sfide significative:

- **Necessità di un numero infinito di episodi**: Nella realtà pratica, dove i tempi di calcolo sono limitati, si accetta comunemente una policy sub-ottima entro una soglia di incertezza ritenuta accettabile.
- **Strategie per l'Esplorazione Continua**:
  1. *Metodi Off-Policy*: Si possono utilizzare due policy separate, una *soft* (stocastica) che guida l'agente nell'interazione con l'ambiente, e una seconda policy che sfrutta le informazioni raccolte dalla prima per effettuare miglioramenti progressivi fino a raggiungere l'ottimalità.
  2. *Metodi On-Policy*: In presenza di una sola policy, questa deve essere *soft*, ovvero soddisfare la condizione:

     $$\text{Soft Policy} \qquad \pi(a|s) > 0 \quad \forall s \in S \quad \text{e} \quad \forall a \in A$$

     Una policy ε-greedy, come esempio di policy ε-soft, può essere utilizzata per mantenere l'equilibrio tra esplorazione ed exploitation.

I metodi Monte Carlo sono particolarmente adatti a contesti con una struttura episodica chiaramente definita, consentendo all'agente di sfruttare l'esperienza completa per prendere decisioni informate. Questi metodi mostrano una notevole robustezza in ambienti caratterizzati da alta varianza e incertezza, grazie alla loro capacità di fornire stime affidabili tramite il campionamento diretto delle interazioni con l'ambiente.

OFF-POLICY MONTE CARLO METHODS

Nei metodi Monte Carlo off-policy, la sfida principale è garantire una sufficiente esplorazione dello spazio delle azioni per identificare la policy ottimale. A differenza dei metodi on-policy, dove tecniche come *exploring starts* o policy ε-greedy possono essere impiegate, i metodi off-policy utilizzano due policy distinte per navigare e apprendere dall'ambiente. Questo approccio permette di separare il processo di esplorazione dalle decisioni che massimizzano i ritorni attesi.

Nei metodi off-policy, le due policy in gioco sono:

1. **Behaviour Policy** - b: Questa policy guida l'esplorazione dell'agente nell'ambiente. È solitamente una policy *soft* (stocastica), assicurando che tutte le azioni possibili vengano esplorate durante gli episodi. La policy b è responsabile di raccogliere dati che saranno poi utilizzati per aggiornare la target policy.
2. **Target Policy** - π: La target policy è quella che l'algoritmo cerca di ottimizzare. Può essere deterministica, spesso orientata verso l'exploitation per massimizzare i ritorni attesi. La target policy viene aggiornata sulla base delle informazioni raccolte seguendo la behaviour policy.

Sebbene i metodi off-policy possano essere più complessi e presentare una varianza più elevata rispetto ai metodi on-policy, offrono il vantaggio di una maggiore flessibilità e potenza. In particolare, permettono all'agente di apprendere da un insieme di esperienze più ampio, potenzialmente raccolto anche da policy diverse da quella attualmente in uso.

Politecnico di Torino

Per questi metodi la fase di *prediction* risulta leggermente differente. Infatti l'algoritmo utilizza l'esperienza effettuata durante l'episodio e sotto la behaviour policy b per creare un update nella target policy π. Per soddisfare la condizione di *coverage*, è necessario che tutte le azioni che potrebbero essere selezionate dalla target policy π siano esplorate dalla behaviour policy b. Matematicamente, questo si traduce in:

$$\pi(a|s) > 0 \quad \Longrightarrow \quad b(a|s) > 0 \qquad \text{Coverage}$$

In questo modo, la behaviour policy deve essere soft, cioè stocastica, consentendo una esplorazione più ampia, mentre la target policy può essere deterministica, spesso una greedy policy che sceglie sempre l'azione con il valore più alto previsto.

Per trasformare le esperienze ottenute seguendo la behaviour policy b in stime utili per il miglioramento della target policy π, i metodi off-policy Monte Carlo utilizzano ciò che è noto come *importance sampling ratio*. Questo rapporto pesa i return ottenuti in base alle relative probabilità delle traiettorie sotto entrambe le policy, consentendo così una stima accurata del valore della target policy, anche quando l'esperienza è stata generata seguendo una policy diversa.

$$\rho_{t:T-1} \doteq \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \qquad \text{Importance Sampling Ratio} \tag{4.23}$$

Dove $\rho_{t:T-1}$ è l'importance sampling ratio dal tempo t al termine dell'episodio $T-1$, $\pi(A_k|S_k)$ è la probabilità di scegliere l'azione A_k in base alla target policy nel contesto dello stato S_k, e $b(A_k|S_k)$ è la probabilità corrispondente in base alla behaviour policy.

Per stimare, ad esempio, lo state value, possiamo utilizzare due approcci:

1. **Ordinary Importance Sampling**: In questo metodo, i return ottenuti vengono riscalati attraverso l'importance sampling ratio e poi mediati. Questo fornisce una stima diretta del valore dello stato:

$$V(S) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} \cdot G_t}{|\mathcal{T}(s)|} \qquad \text{Ordinary Importance Sampling} \tag{4.24}$$

   Dove $\mathcal{T}(s)$ è l'insieme di tutti i tempi in cui lo stato s è stato visitato e G_t è il ritorno totale a partire dal tempo t.

2. **Weighted Importance Sampling**: Questo approccio differisce dall'Ordinary Importance Sampling nel modo in cui i return vengono mediati. Qui, i return vengono pesati in base all'importance sampling ratio, offrendo una stima più bilanciata che riduce la varianza:

$$V(S) \doteq \frac{\sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1} \cdot G_t}{\sum_{t\in\mathcal{T}(s)} \rho_{t:T-1}} \qquad \text{Weighted Importance Sampling} \tag{4.25}$$

Politecnico di Torino

Entrambi i metodi di stima hanno i loro vantaggi e svantaggi. L'Ordinary Importance Sampling è più diretto e intuitivo, ma può soffrire di alta varianza, soprattutto quando il rapporto di importance sampling è grande. Il Weighted Importance Sampling, d'altra parte, tende ad essere più stabile e meno soggetto a variazioni drastiche, rendendolo spesso una scelta preferibile in molti scenari pratici.

EFFICIENZA DEI METODI MONTE CARLO

I metodi Monte Carlo nel Reinforcement Learning offrono un approccio intuitivo e diretto all'apprendimento basato sull'esperienza, ma presentano sfide in termini di varianza delle stime e costo computazionale. Queste sfide si manifestano principalmente nella varianza elevata dei return degli episodi e nella necessità di numerosi episodi per raggiungere stime affidabili.

Per affrontare queste sfide, si possono adottare varie strategie:

- **Riduzione della Varianza**: Tecniche come l'importance sampling, l'utilizzo di baselines, o approcci di bootstrapping possono aiutare a stabilizzare le stime, riducendo la varianza e migliorando l'affidabilità dell'apprendimento.
- **Accelerazione dell'Apprendimento**: Algoritmi che sfruttano più efficacemente i dati raccolti o che riducono la necessità di episodi lunghi possono velocizzare l'apprendimento. L'adozione di metodi off-policy, ad esempio, permette di apprendere da una gamma più ampia di esperienze, potenzialmente riducendo il numero di episodi necessari.

Utilizzando una behaviour policy per l'esplorazione e apprendendo una target policy diversa, i metodi off-policy possono acquisire informazioni in modo più rapido e vario. Questo approccio non solo accelera l'apprendimento, ma può anche fornire stime più precise, migliorando l'efficienza complessiva dei metodi Monte Carlo.

Monte Carlo Algorithm	
On-Policy	- Spazi decisionali: continui e discreti - La policy utilizzata per prendere decisioni è la stessa usata per valutare i return - Tecniche di esplorazione come "exploring starts" o policy ε-greedy - Adatta per situazioni in cui è possibile controllare direttamente l'ambiente di apprendimento
Off-Policy	- Spazi decisionali: continui e discreti - Utilizza una Behaviour Policy per l'esplorazione e una Target Policy per l'apprendimento - Necessaria la condizione di coverage: $\pi(a\|s) > 0 \implies b(a\|s) > 0$ - Uso dell'Importance Sampling Ratio per l'aggiornamento della Target Policy: $\rho_{t:T-1}$ - Applicazioni in scenari dove l'esperienza di apprendimento è raccolta in modo indipendente dalla policy ottimale

Table 4.3: Confronto tra Metodi Monte Carlo On-Policy e Off-Policy

Politecnico di Torino

4.2.4 TEMPORAL-DIFFERENCE LEARNING - TD

Il Temporal-Difference Learning (TD) rappresenta un'innovazione significativa, combinando elementi essenziali sia dei metodi di Dynamic Programming (DP) che di Monte Carlo (MC). Questa metodologia consente di apprendere direttamente dall'esperienza, in modo simile ai metodi MC, integrando al contempo aggiornamenti progressivi delle value function come nei metodi DP. Questo ibrido offre un bilanciamento tra efficienza computazionale e adattabilità alle esperienze in tempo reale.

A differenza dei metodi MC, i metodi TD non aspettano la conclusione di un episodio per aggiornare le value function. Invece, procedono con aggiornamenti incrementali dopo ogni n passi, utilizzando le informazioni attuali in combinazione con stime pregresse, in un processo noto come *bootstrapping*. Questa caratteristica li rende particolarmente adatti per ambienti con episodi lunghi o indefiniti, o in situazioni che richiedono un apprendimento continuo.

Alcuni degli algoritmi TD più rilevanti includono:

- **TD(0)**: Questo metodo rappresenta la forma più elementare di TD Learning. Gli aggiornamenti delle value function si basano sulla differenza tra il ritorno previsto e quello osservato nel passo successivo.
- **SARSA**: SARSA è un algoritmo on-policy che aggiorna le value function basandosi sull'azione corrente e sull'azione futura prevista.
- **Q-Learning**: A differenza di SARSA, Q-Learning è un algoritmo off-policy che mira a imparare la value function di una policy ottimale, indipendentemente dalla policy seguita durante l'esperienza.

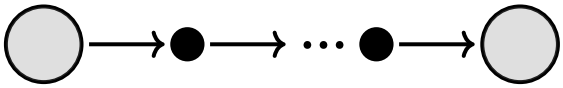


Figure 4.8: n-step TD backup diagram

TEMPORAL-DIFFERENCE PREDICTION

La *Policy Evaluation* nei metodi TD si concentra sulla stima delle value function di una policy data attraverso le differenze temporali, sfruttando quindi le informazioni ottenute attraverso l'esperienza diretta con l'ambiente. A differenza dei metodi Monte Carlo, che richiedono la conclusione di un intero episodio per aggiornare le stime, i metodi TD sfruttano un approccio più dinamico e flessibile, aggiornando le stime dopo ogni n passi temporali.

Questo processo incrementale si basa sulla differenza tra il valore attuale di una stima dello stato e il valore previsto dopo aver osservato l'esito dell'azione successiva. In poche parole, a differenza dei metodi MC, in cui viene usato il return completo (G_t) come *target* per la stima di $V(S_t)$, i metodi TD utilizzano come target la somma dei reward associati agli n time-step successivi.

Politecnico di Torino

In termini matematici, l'aggiornamento TD per uno stato s a un tempo t può essere espresso come segue:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big] \qquad \text{One-Step TD o TD(0)} \tag{4.26}$$

Dove $V(S_t)$ è la stima corrente del valore dello stato S_t, α è il *step-size parameter*, R_{t+1} è la ricompensa osservata dopo aver eseguito un'azione, γ è il fattore di sconto, e $V(S_{t+1})$ è la stima del valore dello stato successivo. Questo metodo di aggiornamento, noto come *Bootstrapping*, sfrutta le stime eseguite in precedenza per calcolare la value function in maniera incrementale. Quindi come per la DP, possiamo calcolare la state value associata ad un singolo stato (s) sotto la policy π attraverso l'equazione 4.17:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi\Big[G_t \mid S_t = s\Big] \\ &= \mathbb{E}_\pi\Big[R_{t+1} + \gamma \cdot G_{t+1} \mid S_t = s\Big] \\ &= \mathbb{E}_\pi\Big[R_{t+1} + \gamma \cdot v_\pi(S_t + 1) \mid S_t = s\Big] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned}$$

Inoltre, come nei metodi MC, seguendo una singola traiettoria durante l'episodio, realizzano ciò che è noto come *sample updates*. Questi aggiornamenti si basano sugli stati osservati lungo la traiettoria, piuttosto che su un'osservazione della distribuzione di probabilità che caratterizzerebbe tutte le possibili traiettorie.

Questo approccio incrementale ha diversi vantaggi:

- **Efficienza in ambienti dinamici**: I metodi TD sono particolarmente adatti per ambienti in cui gli stati cambiano rapidamente o sono continui, poiché consentono aggiornamenti in tempo reale[10] senza dover attendere la conclusione di un episodio.
- **Flessibilità**: La possibilità di aggiornare le stime a ogni n time-step consente ai metodi TD di adattarsi rapidamente alle nuove informazioni, rendendoli efficaci in ambienti complessi e non stazionari.
- **Scalabilità**: A differenza dei metodi Monte Carlo, che possono diventare inefficienti in scenari con episodi lunghi, i metodi TD mantengono la loro efficienza anche in ambienti con orizzonti temporali estesi.

TEMPORAL-DIFFERENCE CONTROL

I metodi TD hanno dato origine a una vasta gamma di algoritmi di controllo, ognuno caratterizzato da specifiche peculiarità e adatto a differenti scenari applicativi. Questi algoritmi per la fase di controllo seguono sempre la *Generalized Policy Iteration* (GPI) e cercano di trovare un trade-off tra exploration ed exploitation.

[10] Spesso in letteratura si utilizza il termine *Online* (oppure *Offline* se gli aggiornamenti non avvengono in tempo reale).

Politecnico di Torino

Tra gli algoritmi più influenti e comunemente utilizzati, troviamo:

1. **SARSA (On-Policy)**:

   L'algoritmo SARSA, acronimo di *State-Action-Reward-State-Action*, è un algoritmo on-policy che aggiorna i valori dell'action value function $Q(s,a)$. L'inizializzazione prevede che $Q(s,a)$ sia impostato arbitrariamente per tutte le coppie stato-azione (s,a), ad eccezione degli stati terminali, dove Q è zero. Una politica, tipicamente ε-greedy, è utilizzata per scegliere le azioni basate sui valori Q attuali.

   Per ogni episodio, si inizia con l'inizializzazione dello stato S e la scelta dell'azione A da S usando la politica basata su Q. Per ogni passo dell'episodio, l'agente esegue l'azione A, osserva il reward R e il nuovo stato S', quindi sceglie una nuova azione A' da S' usando la politica. Il valore $Q(S,A)$ è aggiornato secondo la regola:

   $$Q(S,A) \leftarrow Q(S,A) + \alpha\Big[R + \gamma \cdot Q(S',A') - Q(S,A)\Big] \qquad \text{SARSA}$$

   Dove α è il *learning rate* o *step-size parameter* e γ è il fattore di sconto. Dopo l'aggiornamento, lo stato S e l'azione A vengono aggiornati a S' e A' rispettivamente. Questo processo viene ripetuto per ogni passo fino al raggiungimento di uno stato terminale.

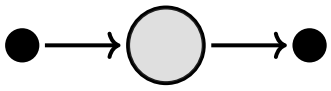
   

   Figure 4.9: SARSA TD on-policy backup diagram

2. **Q-Learning (Off-Policy)**:

   Q-Learning è uno degli algoritmi TD off-policy più noti, che mira a trovare una politica ottimale imparando i valori dell'action value function $Q(s,a)$. L'inizializzazione prevede che $Q(s,a)$ sia impostato arbitrariamente per tutte le coppie stato-azione (s,a), escludendo gli stati terminali, dove Q è zero. Il Q-Learning non segue una specifica politica per scegliere le azioni durante l'apprendimento, ma esplora l'ambiente in modo indipendente.

   In ogni episodio, lo stato iniziale S viene definito. Per ogni passo dell'episodio, l'agente sceglie un'azione A dallo stato S (spesso utilizzando una politica ε-greedy per bilanciare esplorazione ed sfruttamento) e osserva il reward R e il nuovo stato S' dopo aver eseguito l'azione. Successivamente, il valore $Q(S,A)$ viene aggiornato usando la regola:

   $$Q(S,A) \leftarrow Q(S,A) + \alpha\Big[R + \gamma \max_{a'} Q(S',a') - Q(S,A)\Big] \qquad \text{Q-Learning}$$

   Dove α è il *learning rate* e γ è il fattore di sconto. A differenza del SARSA, il Q-Learning aggiorna i suoi valori Q basandosi sul massimo valore stimato di Q nel nuovo stato S', indipendentemente dalla scelta dell'azione effettiva. Questo permette al Q-Learning di imparare la politica ottimale pur seguendo una politica di esplorazione. Lo stato S viene poi aggiornato a S' e il processo continua fino al raggiungimento di uno stato terminale.

3. **Expected SARSA**:

   Expected SARSA è una variante dell'algoritmo SARSA nell'ambito dell'apprendimento per rinforzo, che utilizza le aspettative rispetto alle possibili azioni future per aggiornare la funzione di valore dell'azione, $Q(s,a)$.

   Come nel caso del SARSA, l'inizializzazione avviene impostando $Q(s,a)$ in modo arbitrario per tutte le coppie stato-azione (s,a), escludendo gli stati terminali, dove Q è zero. Expected SARSA utilizza una politica (come la ε-greedy) per determinare il comportamento dell'agente, ma il suo aggiornamento del valore Q differisce dal SARSA tradizionale.

   Durante ogni episodio, lo stato iniziale S viene definito e un'azione A viene scelta da S usando la politica corrente. Per ogni passo dell'episodio, dopo aver eseguito l'azione A, l'agente osserva il reward R e il nuovo stato S'. A differenza del SARSA, che aggiorna Q sulla base dell'azione effettivamente selezionata nel nuovo stato, Expected SARSA considera il valore atteso di Q su tutte le possibili azioni future, ponderato secondo la politica attuale. La regola di aggiornamento è la seguente:

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \sum_{a'} \pi(a'|S')Q(S',a') - Q(S,A)\right] \qquad \text{Expected SARSA}$$

   Dove $\pi(a'|S')$ rappresenta la probabilità di scegliere l'azione a' nello stato S' secondo la politica corrente, α è il *learning rate*, e γ è il fattore di sconto. In questo modo, Expected SARSA incorpora le aspettative future nella sua stima, permettendo un aggiornamento più stabile rispetto al SARSA tradizionale. Lo stato S viene poi aggiornato a S', e il processo si ripete fino al raggiungimento di uno stato terminale.

EFFICIENZA DEI METODI TEMPORAL DIFFERNCE

I metodi Temporal Difference (TD) sono strumenti potenti nel Reinforcement Learning, noti per la loro capacità di apprendere efficacemente da esperienze sequenziali. Tuttavia, l'efficienza di questi metodi può essere limitata da sfide che possono rallentare il processo di apprendimento e influire sulla stabilità della convergenza verso una policy ottimale.

Una delle principali sfide dei metodi TD è la varianza[11] nei loro aggiornamenti, che può portare a un apprendimento instabile e a fluttuazioni significative nelle stime delle value function. Questa varianza è spesso causata dalla natura stocastica degli ambienti di RL e dall'uso di campioni di esperienza limitati per gli aggiornamenti.

In alcuni casi, i metodi TD possono richiedere molti episodi prima di convergere a una soluzione ottimale, specialmente in ambienti complessi con un ampio spazio di stati o azioni. Questo può rendere l'apprendimento inefficace in scenari dove il tempo è un fattore critico.

[11] Nel contesto del RL, il termine *Varianza* si riferisce generalmente a:

1. Varianza dei Return: Si riferisce alla variabilità dei return che un agente riceve in seguito alle sue azioni in diversi episodi o passaggi in uno stesso episodio.
2. Varianza nella stima delle value function: Si riferisce alla variabilità nelle stime delle value function che l'algoritmo sta cercando di apprendere.

Politecnico di Torino

Per affrontare queste sfide e migliorare l'efficienza dei metodi TD, diverse strategie possono essere adottate:

- **Tecniche di riduzione della varianza**: L'adozione di tecniche come *eligibility traces* e *learning rate adaptation* può aiutare a ridurre la varianza negli aggiornamenti TD. *Eligibility traces* combinano i vantaggi dell'apprendimento a breve e a lungo termine, mentre l'adattamento del tasso di apprendimento può aiutare a stabilizzare l'apprendimento in fasi diverse del processo.
- **Approcci off-policy**: L'uso di algoritmi off-policy come Q-Learning e Expected SARSA può migliorare la rapidità e la stabilità della convergenza. Questi metodi separano il processo di esplorazione dalle decisioni di aggiornamento, consentendo agli agenti di apprendere da un insieme più ampio di esperienze e di accelerare l'apprendimento.
- **Integrazione di altre tecniche di apprendimento**: Combinare i metodi TD con altre tecniche di apprendimento, come il Deep Learning, può offrire un equilibrio tra esplorazione, sfruttamento delle informazioni e rapidità di convergenza.

Temporal-Difference Algorithm	
SARSA	- Spazi decisionali: continui e discreti - On-policy - Aggiorna i valori Q in base all'azione corrente e all'azione successiva - Può convergere a una policy sub-ottimale in ambienti con esplorazione insufficiente
Q-Learning	- Spazi decisionali: Efficace in spazi discreti, applicabile anche in contesti continui - Off-policy - Aggiorna i valori Q assumendo sempre la scelta della migliore azione possibile successiva - Tende a trovare policy ottimali garantendo un'adeguata esplorazione
Expected SARSA	- Spazi decisionali: continui e discreti - On-policy o off-policy (a seconda dell'implementazione) - Aggiorna i valori Q basandosi sul valore atteso delle azioni future - Riduce la varianza dell'aggiornamento rispetto a SARSA standard e può migliorare la convergenza

Table 4.4: Confronto tra Metodi Temporal Difference On-Policy e Off-Policy

Politecnico di Torino

MONTE CARLO TREE SEARCH - MCTS

Politecnico di Torino

CHAPTER 5

PLANNING AND LEARNING METHODS

Nel precedente capitolo, abbiamo esplorato i metodi fondamentali di reinforcement learning insieme ai loro corrispondenti algoritmi. Questi approcci possono essere classificati in base alla loro interazione con l'ambiente: *Model Based*, che sfrutta una conoscenza predefinita delle dinamiche ambientali, come evidenziato dagli algoritmi di Dynamic Programming (DP), oppure *Model Free*, che dipende esclusivamente dall'esperienza acquisita tramite l'interazione diretta tra l'agente e l'ambiente. I metodi Monte Carlo (MC) e Temporal Difference (TD) sono esempi di quest'ultima categoria.

Entrambi gli approcci mirano a un obiettivo condiviso: formulare una stima accurata delle *value functions*, sia che si tratti di state-value o di action-value, in funzione dell'algoritmo impiegato.

1. **Model-Based**: In questo paradigma, l'agente ha accesso a un modello dettagliato dell'ambiente e delle sue dinamiche. Questi modelli prediligono un approccio di pianificazione (*Planning*) per determinare le azioni che l'agente dovrà intraprendere durante l'episodio. Tra gli algoritmi principali troviamo:
   (a) Dynamic Programming
   (b) Heuristic Search
2. **Model-Free**: Qui l'agente opera senza un modello definito dell'ambiente e, di conseguenza, senza una comprensione delle sue dinamiche. L'apprendimento avviene attraverso l'esperienza diretta, con l'agente che adatta le sue azioni in base alle risposte ricevute dall'ambiente sotto forma di reward e nuovi stati.

   L'attenzione è focalizzata non sulla pianificazione anticipata ma sull'identificare, tramite l'apprendimento (*Learning*), le azioni più vantaggiose. Esempi notevoli includono:
   (a) Monte Carlo
   (b) Temporal Difference

In entrambi i casi, model-based e model-free, l'ambiente restituisce il reward e la transizione allo stato successivo[1]. La distinzione fondamentale risiede nella consapevolezza dell'agente riguardo alle dinamiche dell'ambiente: possedere un modello dell'ambiente consente di eseguire simulazioni di esperienze prima di agire effettivamente, potenzialmente accelerando la convergenza verso una policy ottimale, benché possa comportare un incremento nel tempo di risposta iniziale.

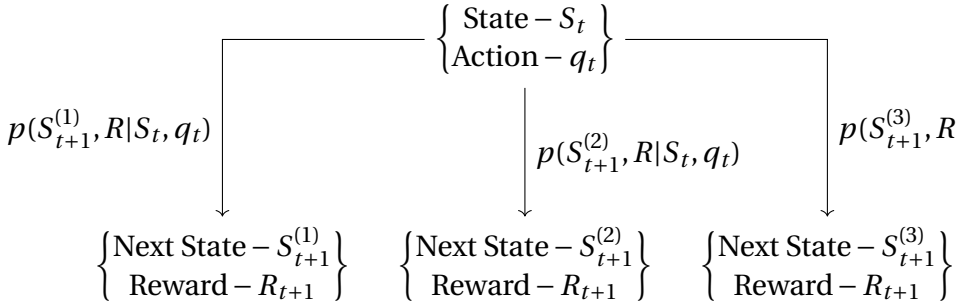
Importante è notare che i modelli dell'ambiente possono essere sia deterministici che stocastici.

$$\begin{Bmatrix} \text{State} - S_t \\ \text{Action} - q_t \end{Bmatrix} \xrightarrow[\textit{Sample Model}]{\text{Environment}} \begin{Bmatrix} \text{Next State} - S_{t+1} \\ \text{Reward} - R_{t+1} \end{Bmatrix} \tag{5.1}$$

[1] In quanto il problema è espresso sotto forma di MDP.

Politecnico di Torino

Un modello deterministico fornisce una singola possibile transizione (*Sample Model*), mentre un modello stocastico presenta una distribuzione di probabilità per le possibili transizioni (*Distribution Model*).



(5.2)

Data una policy specifica (π) e uno stato iniziale (S_t), un *Sample Model* è in grado di generare l'intero episodio, mentre un *Distribution Model* può produrre tutti i possibili esiti e le relative probabilità. Quest'ultimo si rivela generalmente più potente e performante, sebbene richieda maggiori risorse computazionali e sia più complesso da implementare.

In ogni caso, l'uso dei modelli dell'ambiente è indispensabile per simulare scenari e acquisire esperienza simulata, che è fondamentale per l'ottimizzazione delle strategie dell'agente.

5.1 Planning

Il processo di *Planning* capitalizza sul modello dell'ambiente per affinare la policy dell'agente. Ciò si concretizza esplorando lo spazio degli stati per identificare la policy ottimale (*State-Space Planning*) o calcolando le value functions mediante aggiornamenti basati sull'esperienza simulata. Le varianti nei metodi di planning emergono principalmente dal tipo di aggiornamento implementato, dalla sequenza di esecuzione di questi aggiornamenti e dalla persistenza delle informazioni raccolte.

La pianificazione si svolge online, ovvero mentre l'agente interagisce con l'ambiente durante l'episodio. Le nuove informazioni acquisite dall'esperienza diretta con l'ambiente possono essere valorizzate in diversi modi:

1. *Model Learning*: Le informazioni raccolte dall'interazione diretta con l'ambiente sono impiegate per raffinare il modello dell'ambiente, migliorandone l'accuratezza affinché rifletta più fedelmente la realtà.

   È rilevante osservare come l'affinamento del modello ambientale porti, indirettamente, al miglioramento delle value functions e della policy (*Indirect RL*).

2. *Direct RL*: Le informazioni ottenute dall'interazione con l'ambiente reale sono utilizzate per apportare miglioramenti diretti alle value functions e alla policy.

I metodi indiretti tendono a essere più performanti dei metodi diretti, poiché sfruttano in maniera più efficace le informazioni acquisite e riescono a raggiungere una policy superiore anche con un numero ridotto di iterazioni tra l'agente e l'ambiente reale. Tuttavia, il vantaggio dei metodi diretti[2] risiede nella loro semplicità e nell'essere meno suscettibili ai bias del modello dell'ambiente.

[2] I metodi diretti si allineano più strettamente ai processi di apprendimento umano.

Politecnico di Torino

5.1.1 Architettura di Planning

L'architettura di questi modelli si può schematizzare considerando che, in un dato stato S_t durante l'episodio, l'agente interagisce parallelamente sia con l'ambiente reale che col modello dell'ambiente.

Interagendo con il modello dell'ambiente, l'agente sperimenta scenari simulati partendo da uno stato arbitrario e selezionando azioni future casualmente o in base a una policy prestabilita (*search control*). In particolare, può eseguire, a seconda del tipo di modello ambientale, una o più simulazioni dell'episodio e ottenere informazioni che possono essere successivamente impiegate per migliorare le value functions e la policy (*Planning Update*).

Interagendo, invece, con l'ambiente reale, l'agente acquisisce esperienza diretta, che può essere utilizzata per affinare il modello dell'ambiente (*Model Learning* o *Indirect RL*), o per apportare miglioramenti diretti alle value functions e alla policy (*Direct RL Update*).

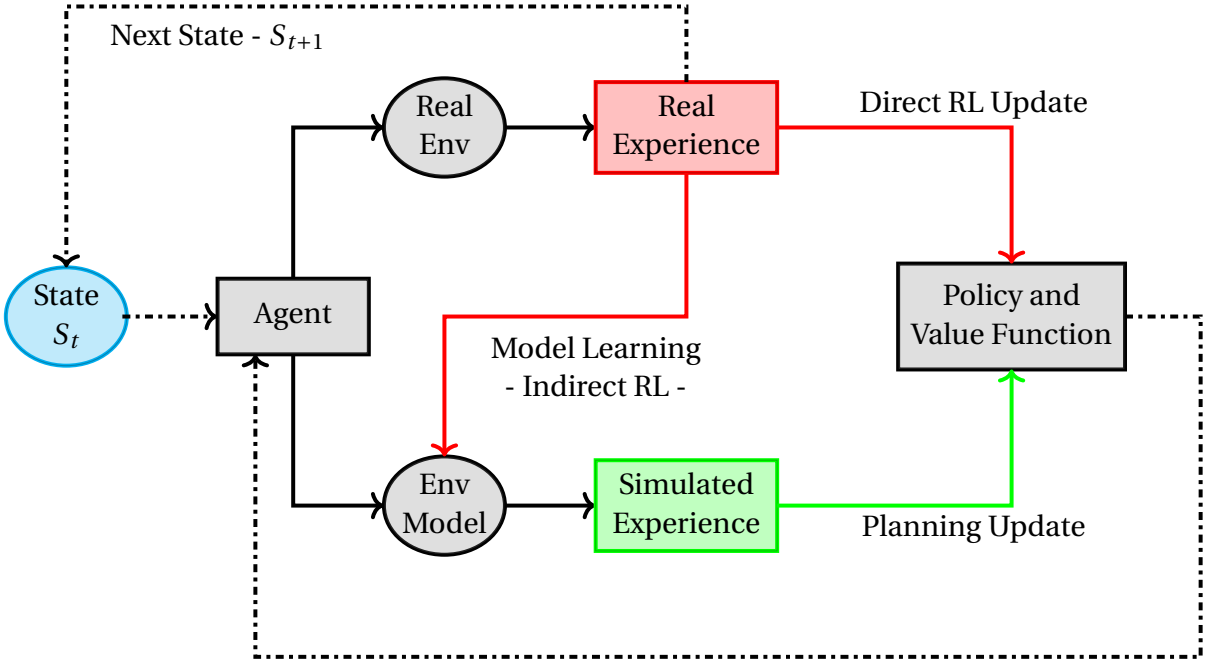


Figure 5.1: Planning Agent Architecture

Gli aggiornamenti e le informazioni vengono elaborati applicando algoritmi di RL sia alle esperienze simulate sia a quelle reali. Generalmente, si predilige l'impiego dello stesso algoritmo per entrambe le tipologie di esperienza.

MODEL ISSUES

L'introduzione di un modello dell'ambiente porta con sé diversi problemi per l'algoritmo. Nonostante gli sforzi di approssimazione accurata, il modello può soffrire di imperfezioni dovute a vari fattori:

1. **Ambiente Stocastico:** La natura intrinsecamente imprevedibile dell'ambiente può introdurre incertezze.
2. **Funzioni Approssimate nel Modello:** L'uso di funzioni approssimative, specialmente nei modelli che incorporano reti neurali (NN) per approssimare le value functions, può portare a discrepanze.
3. **Cambiamenti non Osservati nell'Ambiente:** Variazioni nell'ambiente che non sono state ancora rilevate o incorporate nel modello.

Un modello dell'ambiente afflitto da queste imperfezioni può solo condurre a un planning che, al massimo, produce una policy sub-ottimale. Questo non costituisce generalmente un problema se la policy adottata è ottimistica, tendendo cioè a prevedere un reward maggiore rispetto a quello che verrà effettivamente ricevuto dall'ambiente reale. Tale situazione si verifica quando l'ambiente subisce cambiamenti negativi per l'agente, generando episodi con un return reale inferiore a quello simulato dal modello.

In tali circostanze, l'agente riconosce il cambiamento sfavorevole e si adatta, orientandosi verso una maggiore esplorazione per trovare traiettorie più vantaggiose. Le informazioni accumulate durante questa fase contribuiscono a migliorare il modello dell'ambiente, che si avvicina progressivamente alla realtà effettiva, mitigando il problema dei cambiamenti ambientali.

La situazione è diversa quando l'ambiente evolve in modo da favorire l'agente. In questo caso, l'agente, non più spinto da un'ottimistica aspettativa di reward, continua a seguire la sua policy abituale, che rispecchia le previsioni del modello dell'environment. Questo comporta una ridotta esplorazione, poiché l'agente si percepisce già sulla traiettoria ottimale. In sostanza, l'agente potrebbe non notare il cambiamento favorevole dell'ambiente per un lungo periodo.

Questo scenario evidenzia il conflitto tra exploration ed exploitation nei modelli di planning: l'exploration è essenziale non solo per identificare le migliori policy ma anche per aggiornare e affinare il modello dell'ambiente (Indirect RL).

Idealmente, si desidererebbe che l'agente mantenesse un livello di esplorazione bilanciato per rilevare i cambiamenti ambientali e aggiustare di conseguenza il modello, senza però compromettere le prestazioni globali. Anche qui, si rende necessario trovare un equilibrio adeguato tra exploration ed exploitation.

Politecnico di Torino

Per affrontare questo dilemma, esistono diverse strategie. Una consiste nel monitorare ogni coppia stato-azione visitata durante l'interazione con l'ambiente reale e incrementare il reward simulato in base al tempo trascorso dall'ultima visita.

Questo approccio si basa sull'ipotesi che più tempo passa dalla visita di una specifica coppia stato-azione, maggiore è la probabilità che siano avvenuti cambiamenti significativi. Di conseguenza, si assegna un bonus di reward durante l'esperienza simulata quando l'agente compie tali azioni, calcolato come segue:

$$r^* = r + k\sqrt{\tau} \qquad \begin{cases} r: \text{ Reward originale} \\ k: \text{ Costante di scala piccola} \\ \tau: \text{ Tempo trascorso, espresso in time-steps} \end{cases} \tag{5.3}$$

Questo meccanismo mira a incoraggiare l'esplorazione di parti meno frequentate dello spazio degli stati, aumentando così la robustezza e l'adattabilità della policy.

5.2 Sample and Expected Updates

Gli aggiornamenti delle value function possono essere eseguiti dall'algoritmo utilizzando diverse metodologie, in base alla specifica value function e alla policy che si intende migliorare. Le tipologie di aggiornamenti si suddividono in:

1. **Expected Updates**: Questo tipo di aggiornamento considera tutti i possibili eventi futuri. Prende in considerazione tutte le possibili traiettorie che scaturiscono dallo stato iniziale, fornendo una visione comprensiva delle possibili evoluzioni e dei loro effetti sulla value function. L'aggiornamento previsto, nel caso si volesse calcolare l'action-value, è calcolato come:

$$Q(s,a) = \sum_{s',r} \hat{p}(s',r|s,a)\left[r + \gamma \cdot \max_{a'} Q(s',a')\right] \qquad \text{Expected Update} \tag{5.4}$$

   Una rappresentazione grafica dell'expected update, considerando un singolo time-step può essere la seguente.

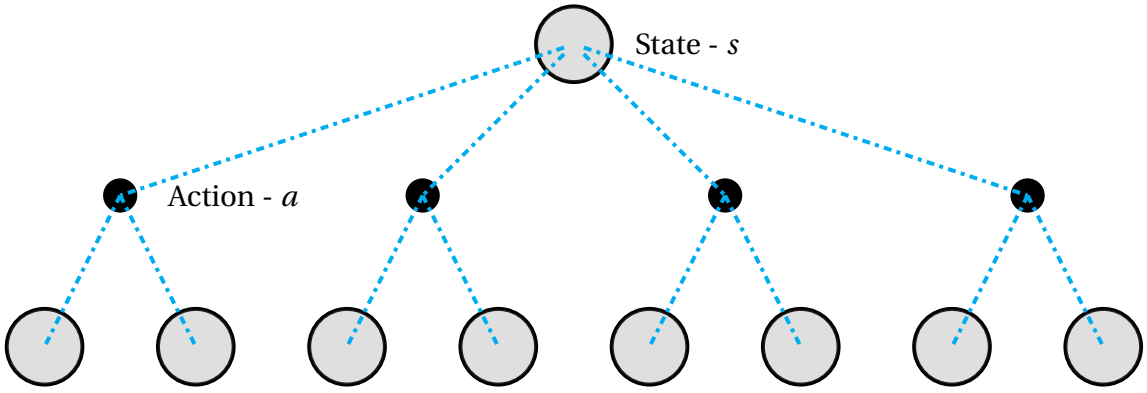


Figure 5.2: Expected Updates

Politecnico di Torino

2. **Sample Updates**: A differenza degli expected updates, questo tipo si basa sull'osservazione di una singola traiettoria (*single sample*), che può essere selezionata casualmente o seguendo una determinata policy. Il sample update è rappresentato dalla formula:

$$Q(s,a) = Q(s,a) + \alpha\left[R + \gamma \cdot \max_{a'} Q(S',a') - Q(s,a)\right] \qquad \text{Sample Update} \qquad (5.5)$$

Una rappresentazione grafica del sample update, considerando un singolo time-step può essere la seguente.

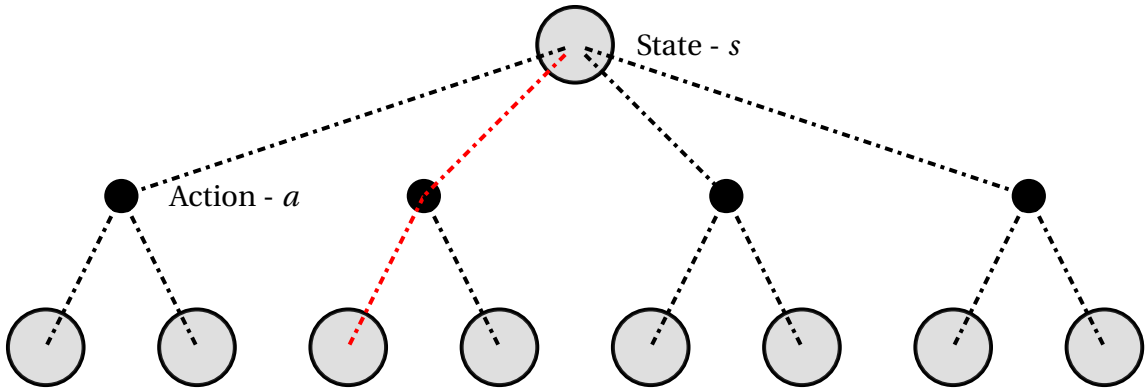


Figure 5.3: Sample Updates

La scelta tra questi due approcci è fortemente influenzata dalla natura del modello dell'ambiente a disposizione. Gli expected updates sono ideali per i distribution models, mentre i sample updates sono più versatili e possono essere applicati in qualsiasi scenario.

Sebbene gli expected updates siano generalmente meno suscettibili agli errori di campionamento e offrano stime più accurate, richiedono un notevole sforzo computazionale poiché considerano tutte le possibili traiettorie. Al contrario, i sample updates, pur essendo soggetti a errori di campionamento, sono computazionalmente più efficienti.

Il fattore di ramificazione (*Branching Factor*, b) gioca un ruolo principale nel determinare il costo computazionale degli aggiornamenti, specialmente per gli expected updates. In scenari con un numero elevato di possibili stati successivi, potrebbe essere più conveniente eseguire numerosi sample updates piuttosto che attendere il completamento di un singolo expected update.

Per determinare l'approccio più adatto al proprio problema, è utile confrontare l'errore di stima, valutato secondo una metrica, in funzione del tempo computazionale associato all'aggiornamento e al branching factor.

Politecnico di Torino

5.2.1 TRAJECTORY SAMPLING

Durante la simulazione degli episodi, le transizioni tra stati possono essere generate casualmente, consentendo così uno *sweeping* uniforme. Questo approccio assicura che tutti gli stati vengano visitati e aggiornati uniformemente ad ogni simulazione, garantendo un'esplorazione completa dell'ambiente.

Tuttavia, un metodo più mirato potrebbe concentrare le transizioni simulate su coppie stato-azione particolarmente rilevanti. In contesti con un elevato numero di possibili stati, calcolare il valore degli stati in modo uniforme lungo l'episodio può risultare inefficace. Gli stati lontani dall'obiettivo del problema, o meno influenzati da variazioni nell'ambiente o da perturbazioni della state value function, tendono a non subire grandi cambiamenti. Di conseguenza, transizioni simulate uniformemente potrebbero valutare stati di scarso interesse e generare aggiornamenti inutili.

Per problemi di grande dimensione, questo approccio generalizzato può significativamente ridurre l'efficienza. È dunque preferibile adottare tecniche che ottimizzino l'utilizzo delle risorse computazionali focalizzandosi sugli stati più promettenti.

1. Il *Backward Focusing* prevede di visitare gli stati che precedono uno stato il cui valore è cambiato, basandosi sul principio che un cambiamento in un dato stato implica potenzialmente variazioni anche negli stati precedenti. Questo metodo indirizza l'attenzione verso gli stati più suscettibili a cambiamenti significativi, migliorando l'efficienza degli aggiornamenti.

   Questo approccio viene ulteriormente raffinato con il *prioritized sweeping*, un metodo che ordina gli aggiornamenti in base alla loro urgenza e priorità. L'algoritmo mantiene una coda per ogni coppia stato-azione, inserendo gli aggiornamenti secondo criteri specifici, basandosi su soglie e valori di priorità diversificati.

2. Il *Forward Focusing* si concentra sugli stati facilmente accessibili dagli stati frequentemente visitati in base alla policy corrente. Questo approccio mira a ottimizzare ulteriormente l'efficienza computazionale, concentrando risorse e attenzione sugli stati che la policy attuale rende più rilevanti e raggiungibili.

Queste strategie di campionamento delle traiettorie mirano non solo a migliorare la velocità con cui l'agente apprende e si adatta ma anche ad aumentare l'efficacia del processo di apprendimento, assicurando che l'attenzione e le risorse computazionali siano focalizzate sulle aree di maggiore interesse e potenziale cambiamento.

Come evidenziato, nel contesto degli sample updates, la traiettoria può essere determinata casualmente (*Sample Uniformly*) o può seguire la policy precedentemente adottata dall'agente. Questo approccio aiuta ad evitare i problemi associati agli expected updates ed è inoltre relativamente semplice da implementare: richiede solamente che l'agente interagisca con il modello dell'ambiente seguendo la policy corrente.

In sintesi, l'esperienza simulata segue questa logica:

1. L'agente, partendo da un generico stato, compie un'azione seguendo la policy corrente.

2. Il modello dell'ambiente reagisce all'azione compiuta, restituendo all'agente il rispettivo reward e determinando la transizione allo stato successivo.

Politecnico di Torino

3. Questi passaggi vengono ripetuti in successione fino alla conclusione dell'episodio, generando così una traiettoria completa.

Ripetendo il processo più volte, si genera una serie di traiettorie, ognuna caratterizzata dalla stessa policy. Questa procedura è nota come *Trajectory Sampling* e si rivela particolarmente efficace nell'ottimizzare l'esperienza simulata per l'agente. Attraverso l'esplorazione ripetuta e guidata, l'agente può affinare progressivamente la sua comprensione dell'ambiente e migliorare la propria policy, adattandosi in maniera efficiente alle dinamiche complesse e variabili del contesto in cui opera.

L'approccio del *Trajectory Sampling* è fondamentale per garantire che l'agente non solo acquisisca una vasta gamma di esperienze attraverso diverse traiettorie ma anche per assicurare che queste esperienze siano pertinenti e utili al miglioramento continuo delle sue decisioni e strategie.

5.3 Heuristic Search & Rollout Algorithm

Gli algoritmi di pianificazione possono operare seguendo due diverse metodologie:

1. ***Background Planning***: Questi algoritmi sfruttano la conoscenza completa delle dinamiche del modello per affinare progressivamente la policy o le value function. Ciò avviene attraverso esperienze simulate che l'agente può effettuare utilizzando il modello dell'ambiente.

   Il vantaggio principale risiede nella capacità di migliorare continuamente la policy, indipendentemente dalle interazioni dell'agente con l'ambiente reale, consentendo un apprendimento costante e spesso più stabile. Questi algoritmi sono particolarmente efficaci in scenari dove la rapidità di risposta dell'agente non è critica, dato che le traiettorie simulate richiedono un certo tempo per essere elaborate.

2. ***Decision-Time Planning***: A differenza della pianificazione in background, questi algoritmi iniziano e completano la pianificazione non appena si verifica una transizione a un nuovo stato. Tale approccio concentra le risorse computazionali sulla decisione imminente, esplorando in modo dettagliato i possibili futuri stati e simulando interi episodi. La scelta dell'azione è dunque informata e ottimizzata sulla base delle traiettorie simulate e dei rispettivi reward, rendendo questo metodo particolarmente adatto in contesti dinamici dove le decisioni devono essere rapide e basate sulle informazioni più accurate disponibili.

Ulteriormente, è possibile integrare le due tipologie di pianificazione, facendole collaborare simultaneamente durante un episodio. Ciò si realizza focalizzando la pianificazione sullo stato corrente (*Decision-Time*) e archiviando i dati ottenuti dalle traiettorie simulate per far sì che possano essere successivamente utilizzati quando l'agente si imbatte in stati futuri già esplorati tramite simulazioni (*Background*).

Heuristic Search

La ricerca euristica è un pilastro per entrambe le metodologie di pianificazione discusse. Gli algoritmi di ricerca euristica mirano a restringere il numero di percorsi da esaminare nell'ambiente, impiegando una funzione euristica per orientare la ricerca verso itinerari apparentemente più vantaggiosi. Questo approccio risulta particolarmente vantaggioso in ambienti caratterizzati da un vasto spazio di stati, dove un'analisi esaustiva sarebbe insostenibile a livello computazionale.

Politecnico di Torino

Nel contesto della ricerca euristica, per ciascuno stato affrontato durante l'episodio, l'algoritmo valuta numerose potenziali diramazioni dell'albero dei possibili stati futuri. Attraverso un'approssimazione del valore delle funzioni di stato, iniziando dai *leaf nodes* alle estremità dell'albero, procede con operazioni di back up[3] fino al nodo corrispondente allo stato attuale. Quindi, viene selezionata l'azione ottimale per lo stato in questione.

Nel *Decision-Time Planning*, la ricerca euristica è fondamentale per individuare con rapidità le azioni più vantaggiose, restringendo l'esplorazione alle traiettorie più probabili di massimizzare il reward. Nel *Background Planning*, invece, essa facilita l'identificazione delle componenti della policy o delle value function che trarrebbero maggior beneficio da approfondimenti e aggiornamenti.

Rollout Algorithm

L'Algoritmo di Rollout rappresenta una strategia avanzata di *Decision-Time Planning*, che si fonda sui principi dei metodi di controllo alla Monte Carlo. Questa metodologia utilizza intensivamente le simulazioni per formulare decisioni ben informate. Il processo inizia analizzando la situazione attuale e procede simulando diverse traiettorie possibili, basandosi sulla policy corrente dell'agente. Dopo aver osservato i risultati di queste simulazioni, l'algoritmo affina la policy, ottimizzando le decisioni future basandosi su un'analisi approfondita delle ricompense potenziali. Ogni simulazione si sviluppa in avanti, "rotolando" fino a un predeterminato orizzonte temporale o fino al naturale termine dell'episodio, accumulando dati preziosi sulle possibili ricompense future e sulle dinamiche dell'ambiente.

A differenza degli algoritmi di controllo alla Monte Carlo, che mirano a perfezionare le value function verso l'ottimalità, l'obiettivo principale dell'Algoritmo di Rollout è di stimare la migliore azione possibile per ogni stato corrente, utilizzando una policy specifica denominata *Rollout Policy*. Questa policy di rollout agisce come un modello temporaneo, guidando le simulazioni per prevedere l'efficacia delle azioni in scenari ipotetici. Nonostante non punti direttamente all'ottimizzazione complessiva delle value functions, l'impatto di questa strategia è significativo: fornisce stime immediate e pragmatiche che sono vitali per la presa di decisioni in tempo reale.

Inoltre, l'Algoritmo di Rollout si distingue per la sua capacità di incorporare conoscenze e tecniche euristiche nel processo decisionale. Attraverso l'uso di una Rollout Policy attentamente selezionata e potenzialmente personalizzata per il contesto specifico, l'algoritmo può sfruttare conoscenze pregresse o strategie euristiche per navigare in modo più efficace l'ambiente complesso. Questo approccio non solo accelera il processo decisionale ma assicura anche che le decisioni prese siano informate dalle migliori prassi e conoscenze disponibili.

Tuttavia, l'efficacia dell'Algoritmo di Rollout è fortemente influenzata dalla qualità della Rollout Policy impiegata. Una policy ben congegnata può significativamente migliorare la precisione delle stime e la qualità delle decisioni prese. Al contrario, una policy inadeguata può portare a valutazioni errate e a scelte subottimali. Pertanto, la selezione e l'eventuale personalizzazione della Rollout Policy sono passaggi critici che richiedono un'attenta considerazione e un'analisi dettagliata.

Politecnico di Torino

[3] I valori di back up possono essere scartati o conservati per affinare ulteriormente la policy e le funzioni di valore.

Politecnico di Torino

CHAPTER 6

MONTE CARLO TREE SEARCH - MCTS

Il Monte Carlo Tree Search (MCTS) è una famiglia di algoritmi di *Decision-Time Planning* che saranno utilizzati in questo elaborato per risolvere il problema in esame.

MCTS adotta metodi alla Monte Carlo per stimare il valore delle mosse a partire dallo stato corrente, calcolando la media dei risultati delle simulazioni di partite o percorsi. Questo algoritmo è iterativo e viene eseguito ogni volta che è necessario determinare una mossa.

L'obiettivo è quello di costruire e raffinare progressivamente un albero decisionale, iniziando dallo stato attuale del gioco, definito come nodo radice (*root node*).

L'algoritmo procede attraverso ripetute fasi di:

1. **Selection**: Partendo dal *root node*, che rappresenta lo stato attuale del gioco (S_t), l'algoritmo segue una policy per l'albero[1], denominata *Tree policy*[2]. Questa politica è adottata per decidere quale percorso intraprendere all'interno della porzione dell'albero contenuta in memoria, bilanciando tra l'approfondimento di nodi con ritorni elevati (exploitation) e l'esplorazione di nodi meno visitati (exploration).

   La fase di selezione si svolge esclusivamente all'interno dell'albero già costruito e prosegue fino a raggiungere un nodo suscettibile di espansione (*leaf node*), ovvero uno stato di gioco non ancora completamente esplorato.

2. **Expansion**: Al raggiungimento di un nodo espandibile, l'algoritmo seleziona una o più azioni non ancora esplorate e aggiunge all'albero uno o più nuovi nodi che rappresentano lo stato successivo del gioco. Questi nuovi nodi vengono memorizzati, e l'albero cresce in dimensioni ad ogni iterazione, servendo come punti di partenza per le simulazioni future.

   Se durante l'espansione si raggiunge uno stato terminale del gioco, l'algoritmo procede direttamente alla fase di backpropagation.

3. **Simulation**: Partendo dall'ultimo nodo aggiunto, l'algoritmo esegue una simulazione, o rollout, che continua il gioco dalla situazione corrente fino alla fine dell'episodio.

   Questa fase sfrutta il metodo Monte Carlo, conducendo le simulazioni solitamente con politiche semplificate, come la selezione casuale delle azioni, per ottenere rapidamente una stima approssimativa del valore dello stato di gioco. In ogni iterazione del MCTS, una sola simulazione viene condotta per ogni espansione.

4. **Backpropagation**: Una volta completata la simulazione, l'algoritmo retrocede lungo il percorso seguito nell'albero, aggiornando i nodi con i risultati ottenuti. Questi aggiornamenti includono solitamente un upgrade del valore degli stati dei nodi ed il numero totale di simulazioni che hanno attraversato ciascun nodo. Le informazioni raccolte sono essenziali per guidare decisioni più informate e accurate nelle iterazioni future dell'algoritmo.

[1] Una policy spesso utilizzata è l'Upper Confidence Bound (UCB), che in questa situazione viene chiamata UCT, ovvero *Upper Confidence for Tree*.

[2] In letteratura, questa è spesso chiamata *Selection Policy [5]*

Politecnico di Torino

Il processo si ripete per un numero prestabilito di iterazioni o fino al raggiungimento di un limite temporale. Ad ogni iterazione, l'albero decisionale si espande e si affina. Al momento di effettuare una mossa effettiva nel gioco, l'algoritmo seleziona il nodo figlio della radice con il miglior valore, basato sulle statistiche raccolte come il numero di vittorie o il punteggio UCB, come prossima azione nel gioco.

In sintesi, MCTS costruisce e migliora continuamente un albero decisionale simulando traiettorie di gioco e utilizzando i risultati per informare scelte future. Le simulazioni aiutano a identificare le mosse più promettenti e a evitare quelle meno vantaggiose, guidando l'algoritmo verso decisioni strategiche informate nel contesto del gioco reale.

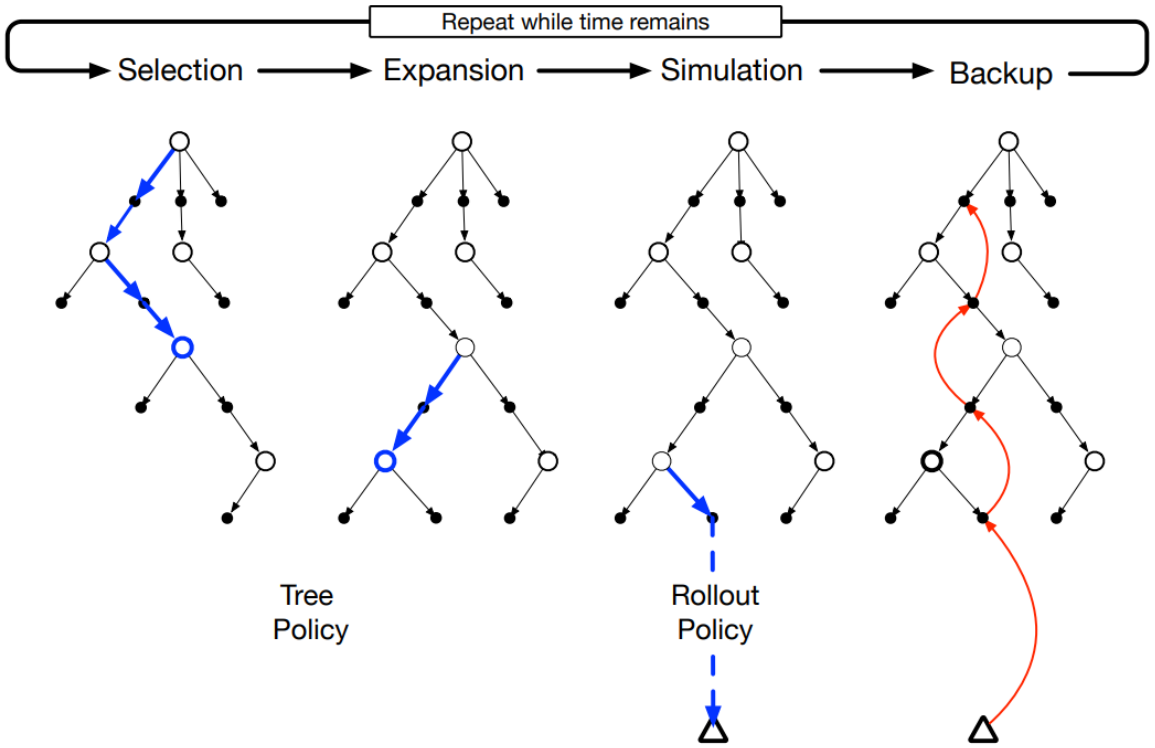


Figure 6.1: Monte Carlo Tree Search, ciclo iterativo. In questa figura sono state adottate le seguenti convenzioni: ○ per rappresentare gli stati e ● per rappresentare le azioni. - [2]

In numerose implementazioni del Monte Carlo Tree Search, quando un'azione viene eseguita nel gioco reale, l'albero decisionale finora costruito non viene scartato completamente. Si adotta piuttosto un approccio di *Tree Reuse* o riutilizzo dell'albero. Dopo che l'azione è stata scelta ed eseguita nel gioco reale, il nodo corrispondente a quella azione diventa il nuovo nodo radice (*root node*). L'algoritmo conserva la parte dell'albero che origina dal nuovo nodo radice, scartando il resto. In altre parole, mantiene tutti i nodi figli e le informazioni raccolte per l'azione appena eseguita, eliminando le parti dell'albero che corrispondono a mosse non intraprese.

Questo approccio ha il vantaggio di preservare informazioni e statistiche preziose accumulate durante le simulazioni precedenti per azioni e stati ancora pertinenti. Ciò può accelerare notevolmente l'apprendimento e il processo decisionale nelle iterazioni future, poiché l'algoritmo non deve ricominciare da capo ma può basarsi sulla conoscenza pre-esistente.

6.1 Extension of Vanilla MCTS Algorithm

L'algoritmo Monte Carlo Tree Search (MCTS) è un *anytime algorithm*, ovvero gode della proprietà di poter essere interrotto in qualsiasi momento, fornendo il miglior valore stimato fino a quell'istante.

Ogni simulazione completata permette all'algoritmo di aggiornare le informazioni sull'albero decisionale, migliorando la qualità della stima per la mossa migliore. Anche in condizioni di limitazione temporale, MCTS è in grado di sfruttare tutte le simulazioni eseguite fino a quel momento per fare una scelta informata, rendendolo uno strumento prezioso in scenari reali dove le decisioni devono essere rapide ma ben ponderate.

L'abilità di MCTS di fornire risultati progressivamente migliori con l'aumentare del tempo di calcolo lo rende particolarmente adatto per una vasta gamma di applicazioni, dalle partite a turni in giochi di strategia, dove il tempo per mossa può essere variabile, fino a sistemi real-time che richiedono risposte immediate. In ogni contesto, MCTS è in grado di bilanciare tra l'esplorazione dell'ambiente di gioco e l'exploitation delle informazioni raccolte, adattandosi efficacemente al tempo disponibile.

Tree Policy

La tree policy (o politica di ricerca) è fondamentale per guidare l'agente nell'albero dei possibili stati precedentemente esplorati, mantenendo un equilibrio ottimale tra l'exploration, ovvero la scelta di azioni meno testate, e l'exploitation, selezionando le azioni con il maggior reward o le azioni migliori. Metodi per implementare una tree policy bilanciata includono:

1. **UCT - Upper Confidence Bounds applied to Trees**:

   L'agente seleziona le azioni seguendo questa formula:

   $$a^* = arg\max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{\ln[N(s)]}{N(s,a)}} \right\} \qquad \text{UCB} \tag{6.1}$$

   La costante C, chiamata *exploration constat*, è decisiva per bilanciare il conflitto tra exploration ed exploitation.

2. **UCB1-Tuned**:

   Questa variante aggiunge alla formula UCB il valore della varianza del punteggio dell'azione a:

   $$a^* = arg\max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{\ln[N(s)]}{N(s,a)} min\left(\frac{1}{4}, \sigma_a + \frac{2 \cdot \ln[N(s)]}{N(s,a)}\right)} \right\} \qquad \text{UCB1-Tuned} \tag{6.2}$$

   Questo approccio mira a un'ulteriore affinazione nell'equilibrio tra exploration ed exploitation, considerando la varianza dei risultati per ogni azione, offrendo una stima più accurata del potenziale di ogni mossa.

Politecnico di Torino

EXPANSION PHASE - RAVE

Nel contesto dell'algoritmo Monte Carlo Tree Search (MCTS), la fase di espansione gioca un ruolo fondamentale nel guidare la ricerca verso decisioni efficaci. Tipicamente, durante questa fase, viene aggiunto un singolo nodo per volta all'albero.

Questa scelta metodologica è il risultato di un compromesso, derivato da esperimenti empirici su una varietà di problemi, che bilancia la velocità di convergenza dell'algoritmo con l'uso ottimale della memoria. L'aggiunta simultanea di più nodi può, infatti, rallentare la convergenza dell'algoritmo e aumentare significativamente il consumo di memoria.

Tuttavia, in certi contesti[3], specialmente quando gli stati esplorati si rivelano appartenere al percorso ottimale, può risultare vantaggioso aggiungere più nodi contemporaneamente. Per massimizzare l'efficienza delle simulazioni e ottimizzare la fase di espansione, è stato introdotto un metodo noto come RAVE (*Rapid Action Value Estimation*). RAVE rappresenta un'evoluzione significativa nell'ambito degli algoritmi MCTS, mirata a migliorare la selezione delle mosse durante la ricerca (*selection*).

Il principio cardine di RAVE è l'ipotesi che certe mosse siano generalmente vantaggiose o svantaggiose indipendentemente dal contesto specifico in cui vengono giocate. Invece di valutare l'efficacia delle mosse esclusivamente nel contesto attuale, come nell'approccio MCTS standard, RAVE accumula e sfrutta le informazioni relative alla frequenza e al successo di ogni mossa attraverso l'intero insieme di simulazioni.

A tal fine, viene mantenuto uno score aggiuntivo denominato Q_{RAVE}, che viene aggiornato per ogni azione eseguita durante la simulazione, non solo per quelle scelte nella fase di selezione.

$$a^* = arg\max_{a \in A(s)} \left\{ \sqrt{\frac{k}{3 \cdot N(s) + k}} \cdot Q_{RAVE}(s,a) + \left(1 - \sqrt{\frac{k}{3 \cdot N(s) + k}}\right) Q(s,a) \right\} \qquad \text{RAVE} \quad (6.3)$$

Dove la costante k viene chiamata *RAVE equivalent constant*.

Durante la fase di selezione dell'albero in MCTS, RAVE introduce un valore aggiuntivo, o bias, alle stime tradizionali basate sul successo delle mosse nel contesto specifico. Questo bias diventa particolarmente influente quando le informazioni su una determinata mossa sono limitate (ad esempio, nelle fasi iniziali della ricerca *cold start*) e si attenua progressivamente con l'accumulo di dati più specifici provenienti dalle simulazioni attuali.

Questo meccanismo permette di orientare la ricerca verso mosse che storicamente hanno mostrato un potenziale promettente, favorendo una convergenza più rapida verso soluzioni di alta qualità.

[3] Nell'implementazione dell'algoritmo MCTS per il problema in esame è stato scelto di espandere tutti tutte le azioni possibili, considerando un massimo di 50 *child nodes*.

Politecnico di Torino

6.2 NEURAL MONTE CARLO TREE SEARCH - NMCTS

Il Neural Monte Carlo Tree Search (Neural MCTS) rappresenta una significativa trasformazione negli algoritmi di *tree search*, per l'elaborazione di strategie nei giochi di intelligenza e nella risoluzione di problemi decisionali complessi. Questa evoluzione segna un salto qualitativo rispetto al tradizionale Monte Carlo Tree Search (MCTS), grazie all'integrazione sinergica con delle DNN.

Le reti neurali vengono integrate in diverse fasi del processo MCTS, ciascuna con un ruolo specifico che migliora l'efficacia generale dell'algoritmo:

1. **Selezione**: La rete neurale, attraverso un processo di *Deep Learning*, acquisisce una comprensione sofisticata dello spazio degli stati e delle azioni possibili. Utilizzando questa conoscenza, assegna una probabilità a priori ad ogni azione in un determinato stato, riflettendo la sua valutazione sulla potenziale efficacia dell'azione.

   Queste probabilità, combinate con strategie euristiche come l'*Upper Confidence Bound* (UCB), forniscono una guida per la selezione delle azioni, bilanciando sapientemente l'esplorazione di nuovi percorsi con l'approfondimento di quelli ritenuti promettenti. Questa sinergia tra intuizione basata sui dati e precisione matematica ottimizza la fase di selezione, dirigendo l'algoritmo verso aree del *tree search* con maggiore probabilità di successo.

2. **Valutazione**: Tradizionalmente, la valutazione dei *leaf node* in MCTS richiedeva l'esecuzione di numerosi roll-out, un processo sia computazionalmente oneroso che spesso impreciso. L'adozione di reti neurali trasforma questa fase, permettendo una stima immediata delle value functions e una distribuzione di probabilità per le future azioni (policy). Questo metodo, basato su un'approfondita comprensione strategica acquisita dalla rete, elimina la necessità di simulazioni prolungate, accelerando significativamente il processo di apprendimento e migliorando la convergenza dell'algoritmo verso soluzioni ottimali.

3. **Backpropagation**: La precisione delle valutazioni effettuate dalla rete neurale sui *leaf node* si riflette direttamente sulla qualità dell'aggiornamento dei valori nei nodi precedenti durante la fase di backpropagation. Utilizzando le stime delle value functions, l'algoritmo aggiorna le metriche di performance dei nodi esplorati, affinando progressivamente la strategia di ricerca. Questo processo iterativo garantisce un miglioramento continuo dell'accuratezza decisionale, rendendo le successive esplorazioni del *tree search* sempre più efficienti e mirate.

4. ***Training* delle reti neurali**: Parallelamente alle varie fasi dell'algoritmo, le esperienze acquisite durante il *planning* vengono utilizzate per il raffinamento delle reti neurali mediante l'addestramento. Questo addestramento, mirato alla minimizzazione della *loss function*, perfeziona le capacità predittive della rete, migliorando la sua abilità nel stimare le funzioni di valore e nell'identificare le politiche ottimali. Attraverso cicli iterativi di esplorazione, valutazione, e addestramento, le reti neurali si evolvono per diventare modelli sempre più accurati e efficienti, capaci di guidare l'algoritmo MCTS verso soluzioni di elevata qualità con una convergenza rapida e robusta.

Il Neural MCTS, quindi, segue un processo logico che integra le capacità delle reti neurali in un quadro di ricerca in albero strutturato. Inoltre, a causa della riduzione o dell'eliminazione delle simulazioni casuali, il Neural MCTS è generalmente più efficiente dal punto di vista computazionale rispetto al suo omologo tradizionale.

Politecnico di Torino

SELEZIONE GUIDATA

Durante la fase di selezione l'agente scende lungo il *tree search*, scegliendo in moto iterativo delle azioni partendo dal *root node*. La scelta delle azioni da eseguire, come nel caso dell'algoritmo MCTS senza l'utilizzo di NN, avviene in base ad una determinata *Tree Policy*, la quale cerca di bilanciare l'*exploitation* e l'*exploration*. Generalmente si utilizzano delle versioni dell'UCB, che prendono la forma seguente:

$$a = \underbrace{\underset{a'}{arg\,max} Q(s,a')}_{Exploitation} + \underbrace{U(s,a')}_{Exploration}$$

Il contributo giocato dalla rete neurale, durante la fase di selezione, ricade nella variazione della componente di esplorazione della *Tree Polocy*. Infatti la rete neurale fornisce una probabilità a priori (*policy*, $P(s,a')$) per ogni possibile azione in uno stato specifico, basata su ciò che ha appreso da esperienze precedenti.

Queste probabilità a priori sono poi combinate all'*Upper Confidence Bound for Tree* (UCT), la strategia euristica utilizzata nell'algoritmo MCTS, in maniera tale da modificare il termine che contribuisce all'esplorazione. La *tree policy* risultante, chiamata PUCT (*Predictor Upper Confidence Bound for Trees*), è un esempio di come l'approccio tradizionale UCB sia modificato per integrare le previsioni della rete neurale.

$$a = \underset{a'}{arg\,max}\; Q(s,a') + \left[c \cdot P(s,a') \cdot \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a')} \right] \qquad \text{PUCT} \tag{6.4}$$

Questo modello di PUCT, ampiamente adottato in ambito di ricerca e implementato in soluzioni come AlphaZero, ha ispirato diverse varianti. Nonostante le modifiche, il principio fondamentale che regola l'equilibrio tra esplorazione ed esploitation rimane costante, attestando l'efficacia di questa metodologia nella guida delle decisioni all'interno dell'albero di ricerca.

ESPANSIONE GUIDATA DALLA RETE NEURALE

L'approccio tradizionale di espansione nel MCTS, che potrebbe considerare uniformemente tutte le possibili azioni da un determinato stato, si rivela spesso impraticabile in domini di gioco o problemi di decisione dove il numero di azioni possibili è vasto o potenzialmente infinito. In queste circostanze, l'adozione di una strategia di espansione non filtrata[4] porterebbe a un'esplorazione inefficiente dello spazio di ricerca, con un conseguente aumento dei tempi computazionali e una diminuzione della qualità delle decisioni prese.

Inserendo una rete neurale in questo processo, l'espansione diventa un'operazione mirata. La rete, addestrata su dati di gioco o su scenari decisionali specifici, apprende a riconoscere le caratteristiche che rendono un'azione particolarmente vantaggiosa o, al contrario, svantaggiosa. Attraverso questa apprendimento, la rete neurale è in grado di fornire una valutazione a priori (*policy*) di ciascuna azione possibile in uno stato dato, assegnando a ciascuna un punteggio di priorità basato sulla sua valutazione del potenziale di successo.

[4] Nel problema in esame le azioni saranno mascherate, in modo da considerare solo quelle ingegneristicamente rilevanti.

Politecnico di Torino

Durante la fase di espansione, quindi, anziché generare tutti i possibili nodi successori di uno stato, l'algoritmo Neural MCTS utilizza le previsioni della rete neurale per selezionare un sottoinsieme ristretto di azioni che sembrano più promettenti. Questo approccio permette un'allocazione più efficiente delle risorse computazionali, focalizzando l'attenzione e i calcoli su quelle parti dell'albero di ricerca che hanno maggiori probabilità di condurre a risultati positivi.

VALUTAZIONE GUIDATA

In questa fase, la rete neurale gioca un ruolo centrale, sostituendo i tradizionali roll-outs casuali con valutazioni dirette e basate su dati. La funzione $f_\theta(s)$, tipicamente una *Deep Neural Network* (DNN) addestrata su vasti set di dati, esegue questa valutazione e produce due output fondamentali:

1. **Stima del Valore dello Stato** ($v(s)$): Questo valore è una stima del potenziale o della qualità di uno stato specifico, che può rappresentare, ad esempio, la probabilità di vittoria in un gioco partendo da quel particolare stato. Questa stima è cruciale per guidare la ricerca verso stati promettenti. La precisione di $v(s)$ è migliorata attraverso l'apprendimento continuo, con la rete neurale che affina le sue previsioni basandosi su risultati di giochi precedenti o simulazioni.
2. **Distribuzione di Probabilità sulle Mosse Future** ($p(s,a)$): La rete neurale fornisce anche una distribuzione di probabilità che valuta la qualità strategica di ciascuna possibile mossa dallo stato attuale. Questo permette all'algoritmo di ponderare le mosse potenzialmente vantaggiose più pesantemente durante la fase di selezione, assicurando che le mosse con maggiori probabilità di successo ricevano maggiore attenzione.

La funzione $f_\theta(s)$ è progettata con l'obiettivo primario di massimizzare le sue capacità predittive riguardo l'esito di un problema, basandosi sullo stato attuale s. Questa ottimizzazione si realizza attraverso un processo di addestramento che affina progressivamente la funzione, permettendole di stabilire correlazioni efficaci tra gli stati osservati e gli esiti corrispondenti.

In questo contesto, l'algoritmo si configura come un'istanza di *Supervised Learning*, il quale si distingue per la sua capacità di operare a partire da un insieme di dati specificamente etichettati. Questo dataset target è costituito dalle esperienze accumulate attraverso iterazioni successive tra l'agente e il modello dell'ambiente in cui opera. Tale dinamica sottolinea l'importanza di un feedback diretto nel processo di apprendimento, dove ogni esempio nel dataset funge da punto di riferimento per la valutazione delle prestazioni e l'aggiustamento dei parametri della funzione $f_\theta(s)$.

La peculiarità dell'approccio risiede nella sua natura ibrida: al cuore di un algoritmo di *Reinforcement Learning*, volto all'ottimizzazione di una politica decisionale attraverso la valutazione di ricompense e penalità derivanti dalle interazioni con l'ambiente, si trova annidato un meccanismo di apprendimento supervisionato.

Questa integrazione consente di sfruttare i punti di forza di entrambe le metodologie: da un lato, l'apprendimento supervisionato fornisce una base solida per la modellazione delle relazioni causa-effetto tra azioni e risultati, dall'altro, il RL promuove un adattamento continuo alle variazioni dell'ambiente e agli obiettivi di lungo termine.

Politecnico di Torino

6.2.1 Addestramento della Rete di Guida in MCTS

Gli approcci per l'addestramento di queste reti sono diversi e possono essere implementati in vari modi a seconda delle specifiche esigenze dell'applicazione.

Un approccio comune è quello di addestrare la rete neurale durante l'esecuzione dell'algoritmo MCTS utilizzando i dati generati dalle simulazioni[5]. In questo scenario, la rete apprende progressivamente man mano che l'algoritmo esplora lo spazio di ricerca. Questo metodo è particolarmente vantaggioso in ambienti dove la conoscenza precedente è limitata o dove le dinamiche sono complesse e difficili da modellare. Il processo di addestramento tipicamente comporta l'aggiornamento dei pesi della rete basato sui risultati delle simulazioni. In letteratura ci si riferisce a questa procedura, ovvero utilizzare l'algoritmo MCTS per migliorare la policy, come *Policy improvement operator*.

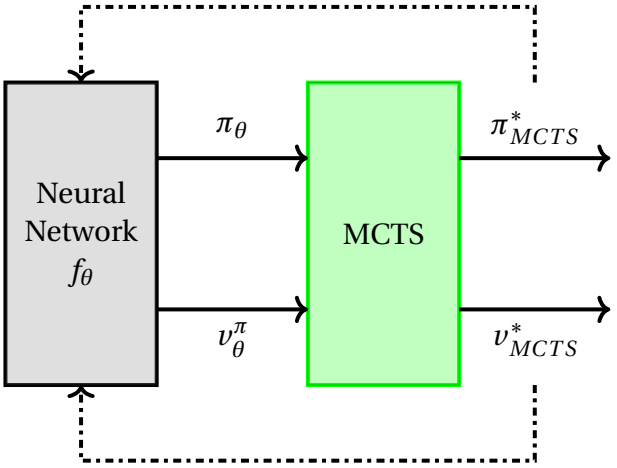


Figure 6.2: *Policy improvement operator*. La *learned policy* (π_θ) restituita dalla NN guida l'agente in diverse fasi dell'algoritmo MCTS. Questa ricerca guidata porta allo sviluppo di una nuova policy (π_{MCTS}) per il *root node*, generalemnte migliore della policy precedentemente predetta dalla NN. L'errore di predizione della NN può quindi essere sfruttato per addestrare la rete neurale. - [3].

Un'alternativa prevede l'addestramento della rete neurale prima del suo utilizzo nell'algoritmo MCTS[6]. In questa fase di pre-addestramento, la rete è tipicamente addestrata su un vasto dataset che rappresenta lo spazio del problema. Questo dataset potrebbe comprendere dati storici, mosse di esperti nel contesto dei giochi, o dati simulati generati con altri mezzi. Il vantaggio di questo approccio è che la rete entra nel processo MCTS con una conoscenza già stabilita, che può accelerare significativamente il processo decisionale. Tuttavia, la limitazione è che il modello pre-addestrato potrebbe non adattarsi bene a scenari non visti o potrebbe essere influenzato dai dati su cui è stato addestrato.

Un approccio ibrido, spesso definito come *warm-start*, combina sia il pre-addestramento che l'addestramento in situ. Qui, la rete neurale è inizialmente addestrata su un dataset preesistente e poi ulteriormente raffinata durante il processo MCTS. Questo approccio mira a bilanciare i benefici di avere un modello pre-addestrato con l'adattabilità fornita dall'apprendimento continuo. Nel *warm-start*, l'addestramento iniziale fornisce un livello di base di conoscenza e strategia, mentre la formazione continua durante l'esecuzione MCTS consente alla rete di adattarsi e ottimizzare le sue previsioni basate sui dati correnti.

[5] Questo è l'approccio adottato in AlphaZero.

[6] L'algoritmo precedente ad AlphaZero, chiamato AlphaGo, utilizzava questa implementazione.

Politecnico di Torino

APPLICATION OF MCTS IN A TRAJECTORY OPTIMISATION PROBLEM

Politecnico di Torino

Politecnico di Torino

CHAPTER 7

SPACE TRAJECTORY OPTIMISATION PROLEM

7.1 GTOC 11: *Dyson Sphere Bulding*

La Global Trajectory Optimisation Competition (GTOC) 11 presenta una missione concentrata sulla progettazione e costruzione di una "Sfera di Dyson" attorno al Sole. Il concetto, inizialmente proposto da Freeman Dyson nel 1960, immagina la creazione di una vasta struttura artificiale, costituita da una miriade di satelliti che, orbitando avvolgono il Sole, e catturano una quantità massiccia della sua energia.

La missione sfrutta avanzamenti tecnologici ipotetici, prevedendo che nel prossimo secolo la propulsione spaziale subirà miglioramenti significativi e sarà sviluppato un dispositivo multifunzionale, chiamato *asteroid transfer device* (ATD), in grado di poter manovrare corpi celesti come asteroidi. L'ATD può essere impiegato su un asteroide per convertire i suoi materiali in propellente, fornendo una spinta continua e consentendo manovre spaziali sofisticate.

Attraverso l'utilizzo di questo dispositivo futuristico, l'obiettivo della missione è costruire un *Dyson Ring* preliminare, organizzato secondo un'insieme di stazioni solari posizionate in un'orbita circolare attorno al Sole, utilizzando materiali derivati da asteroidi per la loro costruzione. La task della competizione consiste nel progettare l'orbita del Dyson Ring, posizionando 12 stazioni in essa e ideando una serie di missioni per trasferire gli asteroidi verso queste stazioni, con l'obiettivo di massimizzare la massa degli asteroidi trasferiti e minimizzare il costo (propellente) utilizzato per tali missioni.

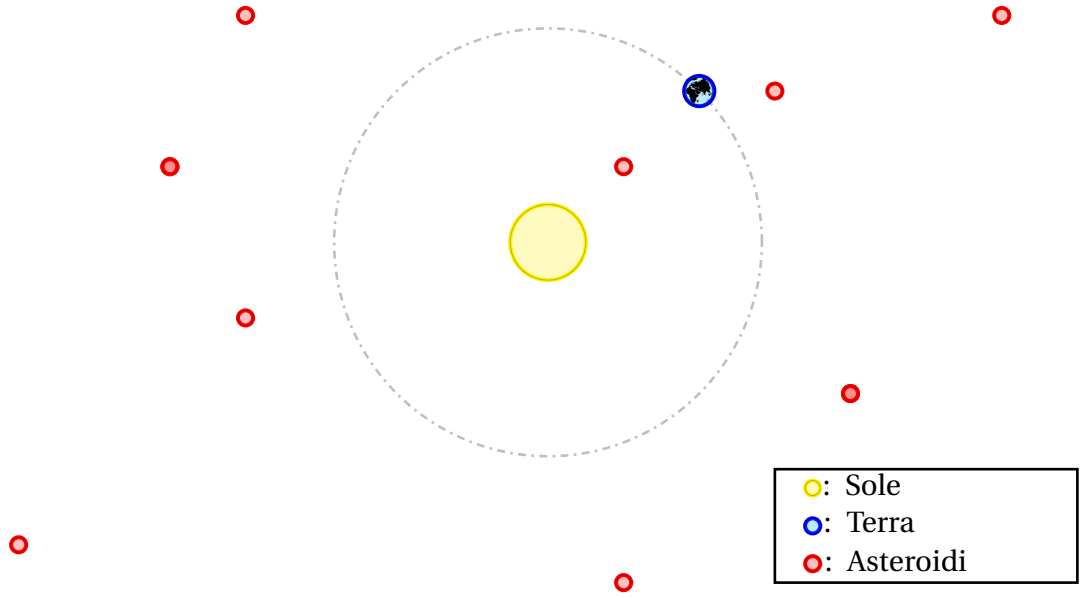


Figure 7.1: Posizione iniziale di Terra ed Asteroidi.

L'essenza di questo progetto risiede nell'utilizzo di materiali estratti da asteroidi, selezionati da un vasto insieme di oltre 83000 candidati[1], per costruire tali stazioni. L'obiettivo è duplice: da un lato, massimizzare la massa complessiva degli asteroidi trasferiti alle stazioni, e dall'altro, minimizzare il costo in termini di propellente, consumato dalle *Mother Ships* durante le rispettive trasferte nella fase preliminare della missione ed utilizzato dal dispositivi ATD nella fase successiva della missione.

[1] I quali parametri orbitali sono descritti nel file `Candidate_Asteroids.txt` - A

Politecnico di Torino

FASE DI RILASCIO DEGLI ATD

La prima fase della missione prevede un'ambiziosa operazione di lancio di massimo 10 *Mother Ships* dalla Terra, ognuna con un eccesso iperbolico di velocità variabile tra 0 e 6 km/s.

$$0 \leq v_\infty \leq 6 \quad [km/s]$$

Le *Mother Ships* possono essere lanciate in maniera totalmente indipendente. Inoltre la finestra di lancio per queste ultime deve essere tra il 1 Gennaio 2121 00:00:00 UT (MJD 95739) ed il 1 Gennaio 2141 00:00:00 UT (MJD 103044) incluso. Una volta lasciata l'orbita terrestre, si dedicano a una serie di manovre impulsive e a randezvous di asteroidi. Durante questi randezvous, ciascuna *Mother Ships* rilascia un *asteroid transfer device* (ATD), progettato per catturare e trasferire gli asteroidi selezionati verso le stazioni di costruzione del Dyson ring.

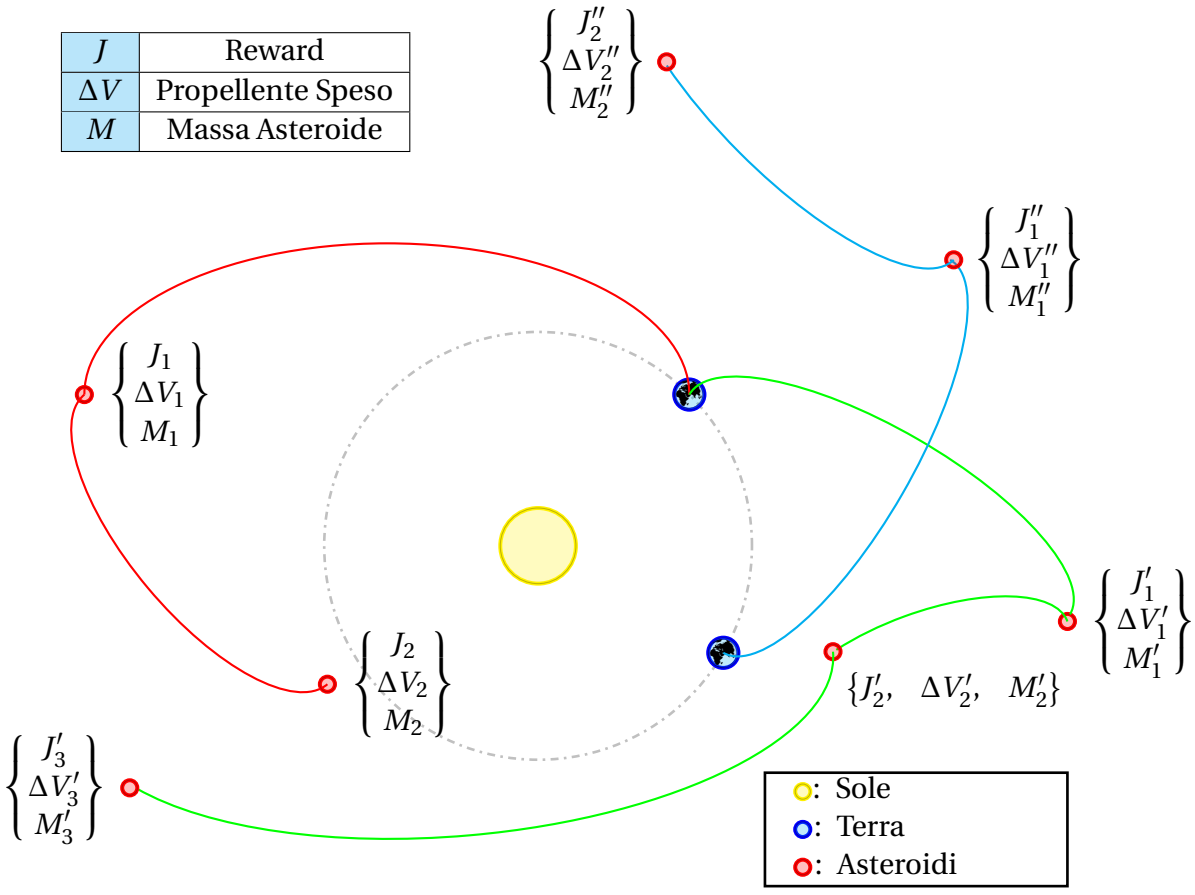


Figure 7.2: Esempio di possibili traiettorie di 3 *Mother Ships*.

Le *Mother Ships*, per poter rilasciare il dispositivo ATD durante i randezvous, sono vincolate ad avere una velocità relativa con l'asteroide strettamente inferiore a 2 km/s. Inoltre il numero di manovre impulsive effettuabili durante ogni trasferta è limitato[2] ad un massimo di 4. Questa fase della missione è caratterizzata da una complessa pianificazione di traiettorie, mirando a ottimizzare la massimizzazione della massa totale degli asteroidi catturati, e nel contempo a minimizzare il ΔV.

[2] Il codice relativo alle dinamiche dell'ambiente trascurerà questo vincolo, in quanto il codice è strutturato per ammettere una singola manovra impulsiva tale da permettere alla *Mother Ships* di raggiungere l'asteroide target della trasferta.

Politecnico di Torino

FASE DI TRASFERIMENTO DEGLI ASTEROIDI

La seconda fase della missione è caratterizzata dall'uso dei dispositivi ATD per manovrare gli asteroidi, catturati dalle *Mother Ships* nella fase precedente, verso la costruzione del Dyson ring. Le manovre sono cruciali per il posizionamento preciso degli asteroidi nelle orbite designate.

Gli ATD, attivati su ciascun asteroide, forniscono la spinta che permette la manovra fino all'arrivo alla rispettiva stazione del Dyson ring. Le manovre degli asteroidi, sono modellate come manovre a spinta continua con una accelerazione fissa:

$$\Gamma_{\text{ATD}} = 1 \times 10^{-4} \quad m/s^2$$

A causa dell'uso dell'ATD, la massa dell'asteroide diminuisce con un flusso di massa proporzionale alla sua massa iniziale:

$$\begin{cases} m^{ast}(\Delta t) = m_0^{ast} - \dot{m} \cdot \Delta t \\ \dot{m} = \alpha \cdot m_0 ast \end{cases}$$

Dove abbiamo espresso con α il coefficiente di proporzionalità:

$$\alpha = 6 \times 10^{-9} \quad s^{-1}$$

Questa dinamica è essenziale per determinare la massa residua dell'asteroide all'arrivo alla stazione di costruzione e per ottimizzare l'uso della spinta fornita dall'ATD.

Il semiasse maggiore (a_{Dyson}) dell'orbita circolare[3] selezionata per la costruzione dell'anello è un parametro chiave, che deve essere ottimizzato per assicurare la massima efficienza e il minor consumo di risorse. L'obiettivo è bilanciare ottimamente il semiasse maggiore dell'orbita e la massa degli asteroidi trasportati, tenendo conto delle limitazioni temporali e delle capacità propulsive degli ATD.

La funzione obiettivo (*Performance Index* - J) della missione è definita come:

$$J = \frac{10^{-10} \cdot M_{\min}}{a_{Dyson}^2 \cdot \sum_{k=1}^{10} \left(1 + \frac{\Delta V_k^{\text{tot}}}{50}\right)^2} \qquad \text{Performance Index} \tag{7.1}$$

Dove $M_{\min}$ è la massa minima tra tutte le stazioni di costruzione, e ΔV_k^{tot} rappresenta l'incremento totale di velocità della k-esima *Mother Ship*. Questa formula mira a massimizzare la massa degli asteroidi trasferiti minimizzando al contempo il consumo complessivo di propellente. L'indice di prestazione J enfatizza l'importanza di una costruzione efficiente del Dyson ring, promuovendo soluzioni che utilizzano in modo ottimale le risorse e minimizzano l'impatto energetico delle manovre spaziali.

[3] Questo è un requisito della missione. Infatti abbiamo libera scelta sui quattro parametri orbitali di:

1. a - Semiasse maggiore (non inferiore a 0.65 AU)
2. i - Inclinazione
3. Ω - Longitudine del nodo ascendente (RAAN)
4. ϕ_i - Fase della prima stazione costruita.

Politecnico di Torino

7.2 Environment Dynamic Model

Il modello dinamico adottato assume una rappresentazione semplificata del moto nello spazio, basata sui principi del moto Kepleriano. La dinamica dell'ambiente si basa quindi sul problema dei due corpi, ovvero un modello matematico, che descrive il movimento di due punti materiali che interagiscono solo attraverso la forza di gravitazione universale.

Il problema dei due corpi è descritto dalla seguente equazione differenziale, che esprime il movimento di un corpo in orbita sotto l'influenza gravitazionale di un altro corpo:

$$\ddot{\mathbf{r}} + \frac{\mu}{r^2}\hat{r} = 0 \qquad \text{Equazione del Moto}$$

Definiamo il parametro gravitazionale μ, come:

$$\mu = GM \quad \left[\frac{km^3}{s^2}\right] \qquad \begin{cases} \mu_\oplus = 3.98600 \cdot 10^5 \left[\frac{km^3}{s^2}\right] \\ \mu_\odot = 132712.44 \cdot 10^6 \left[\frac{km^3}{s^2}\right] \end{cases}$$

Sotto questa premessa, le posizioni e le velocità delle navi spaziali e degli asteroidi sono rappresentate all'interno del sistema di coordinate eliocentrico eclittico J2000, denotate rispettivamente con $\vec{r}$ (posizione) e $\vec{v}$ (velocità).

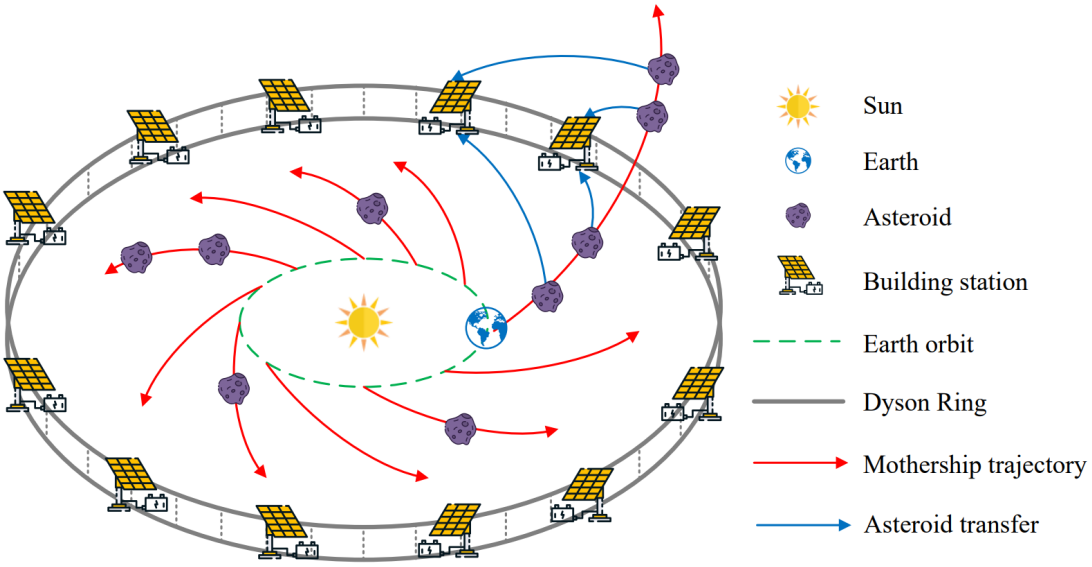


Figure 7.3: Illustrazione delle varie fasi della missione - [4]

Il modello dinamico è formulato attraverso un insieme di equazioni differenziali, che descrivono la variazione della posizione e della velocità nel tempo, così definite:

$$\begin{cases} \dot{r} = v \\ \dot{v} = -\frac{\mu_\odot}{r^3}\hat{r} \end{cases} \tag{7.2}$$

Dove $\mu_\odot$ rappresenta il parametro gravitazionale del Sole.

Politecnico di Torino

Il modello del problema dei due corpi si fonda sull'ipotesi che il moto di un asteroide sia influenzato principalmente dalla forza gravitazionale esercitata da un unico corpo massivo, in questo caso, il Sole. Questo approccio semplificato consente di descrivere l'orbita di un asteroide attraverso equazioni che ammettono soluzione analitica, le quali considerano soltanto la mutua attrazione gravitazionale tra l'asteroide e il Sole.

Questa teoria ci permette di calcolare le orbite future degli asteroidi a partire da un insieme di dati osservativi raccolti in un determinato momento (MJD 59396 ovvero il giorno 1 Luglio 2021). Una volta determinati i parametri orbitali iniziali, come posizione e velocità, il modello consente di propagare l'orbita nel tempo, prevedendo dove l'asteroide si troverà in momenti futuri. Questa osservazione iniziale degli asteroidi e dei rispettivi parametri orbitali è racchiusa nel file `Candidate_Asteroids.txt`, presente in parte in appendice.

L'adozione di questa semplificazione permette di delineare con precisione le orbite degli asteroidi per l'intera durata della missione di interesse. Attraverso l'impiego di un modello orbitale semplificato, basato sul moto Kepleriano, siamo in grado di sviluppare una rappresentazione dell'ambiente spaziale che, pur presentando una certa semplificazione, risulta estremamente accurata e predittiva.

É importante sottolineare che, nonostante il modello adottato riduca la complessità del contesto spaziale, esso offre una rappresentazione deterministica dell'ambiente, eliminando gran parte delle incertezze. In questo scenario specifico, dove si assume un moto puramente Kepleriano, l'agente opera in un ambiente le cui dinamiche sono teoricamente note a priori.

Il modello Kepleriano, pur essendo una semplificazione, è estremamente efficace nel descrivere il moto orbitale in condizioni ideali. Esso costituisce la base per la progettazione delle traiettorie, permettendo al modello di calcolare le orbite iniziali e di pianificare le manovre necessarie per l'incontro con gli asteroidi e la successiva costruzione del Dyson ring.

Sebbene in diversi contesti sia imprescindibile considerare una serie complessa di fattori dinamici, quali le perturbazioni gravitazionali esercitate da altri corpi celesti, gli effetti non-kepleriani dovuti a forze quali la pressione di radiazione solare, e le intrinseche limitazioni dei sistemi di propulsione disponibili, il modello kepleriano offre una base di partenza fondamentale.

Nel contesto specifico della Global Trajectory Optimisation Competition (GTOC), il focus principale non si colloca sulla precisione assoluta della modellizzazione astronomica, bensì sulla capacità di testare e sviluppare algoritmi di ottimizzazione avanzati per la progettazione di traiettorie spaziali.

Politecnico di Torino

7.2.1 Funzioni della Classe dell'Ambiente

Selezione degli Asteroidi

La ricerca di una soluzione ottimale per le traiettorie delle *mother ships* implica la necessità di ridurre le dimensioni di un problema caratterizzato da uno spazio decisionale ampio.

Uno degli approcci primari per raggiungere questo obiettivo consiste nella selezione di un sottoinsieme di asteroidi dai 83453 possibili, come indicato nell'appendice A (Candidate_Asteroids.txt). Questo processo di selezione mira a identificare un gruppo di asteroidi che siano ingegneristicamente più rilevanti per la missione.

La selezione degli asteroidi considerati durante una simulazione si basa sui seguenti criteri principali:

1. **Range di Inclinazione**: La prima fase di scrematura viene effettuata considerando un range di inclinazione orbitale rispetto all'eclittica. Questa scelta è motivata dal fatto che le manovre per il cambio di piano orbitale sono generalmente le più onerose in termini di ΔV. Limitando le inclinazioni orbitali degli asteroidi, si riduce la necessità di manovre complesse e costose, rendendo il problema più gestibile.
2. **Numero di Asteroidi**: Dopo aver filtrato gli asteroidi in base all'inclinazione orbitale, si procede alla selezione di un numero n di asteroidi, basandosi sulla loro massa. Tale criterio è guidato dall'importanza che la massa degli asteroidi riveste nel *performance index* 7.1 della missione. Infatti, il contributo degli asteroidi meno massivi al *performance index* risulterebbe trascurabile, anche se le *motherships* transitassero nelle loro immediate vicinanze.

L'adozione di questi criteri di selezione consente di concentrare le risorse computazionali e gli sforzi di pianificazione su un sottoinsieme di asteroidi che sono più probabilmente vantaggiosi per il raggiungimento degli obiettivi della missione. Questo approccio riduce significativamente la complessità dello spazio decisionale, aumentando l'efficienza della ricerca di traiettorie ottimali e della simulazione complessiva.

Action Masking

Utilizzando uno spazio decisionale discreto per l'ambiente è utile ridurre il numero di azioni possibili in un determinato stato, selezionando solamente un sottoinsieme di azioni promettenti. Questa strategia viene spesso effettuata nell'ambito del reinforcement learning e della pianificazione in ambienti con spazi d'azione estremamente vasti. Tale approccio, noto come *action masking* o mascheramento delle azioni, è fondamentale per gestire efficacemente la complessità in scenari decisionali ad alta dimensionalità.

Nei contesti come quello della nostra missione, dove lo spazio delle azioni è particolarmente ampio, l'enumerazione e la valutazione di tutte le azioni possibili diventano computazionalmente proibitive. Molte delle azioni in un ampio spazio potrebbero essere irrilevanti o controproducenti per il raggiungimento dell'obiettivo.

La selezione di un sottoinsieme di azioni promettenti consente di concentrare le risorse computazionali e l'attenzione su quelle azioni che hanno maggiori probabilità di essere ottimali o vicine all'ottimalità. Questo approccio può essere implementato in vari modi, ad esempio attraverso l'uso di euristiche basate sulla conoscenza del dominio.

Politecnico di Torino

Sono state quindi implementate maschere di azione basate su specifici vincoli di velocità relativa rispetto al vettore velocità dell'asteroide target. Partendo da uno stato iniziale S_t, il numero di azioni permesso per le *motherships* corrisponde al numero di asteroidi che si qualificano come raggiungibili entro i vincoli temporali e di velocità impostati.

Gli asteroidi considerati raggiungibili sono quelli che possono essere intercettati entro un tempo di trasferta interplanetaria prestabilito, con il calcolo della traiettoria ottimale ottenuto attraverso la soluzione del problema di Lambert. Il problema di Lambert, che si occupa di determinare la traiettoria orbitale conica che collega due punti nello spazio in un tempo definito, è risolto iterativamente per diversi intervalli di tempo di trasferta, selezionando come punto di partenza un intervallo minimo di 90 giorni e incrementando questo valore di 60 giorni per ogni iterazione successiva.

La risoluzione del problema di Lambert consente di calcolare il valore del ΔV necessario per l'impulso della manovra e di calcolare la velocità della *motherships* al termine della trasferta. La conoscenza della velocità finale della *motherships* e della velocità dell'asteroide permette di calcolare la velocità relativa tra i due corpi, permettendo di generare delle maschere considerando i seguenti intervalli di velocità relative e stabilendo così dei limiti precisi entro i quali devono rientrare le manovre per essere considerate valide.

$$2\ km/s \quad - \quad 3\ km/s \quad - \quad 4\ km/s \quad - \quad 5\ km/s$$

REWARD FUNCTION

La *reward function* utilizzata nell'algoritmo di reinforcement learning si differenzia dal *Performance Index* 7.1 precedentemente descritto. Questa distinzione è dovuta al focus specifico della fase preliminare, che è quello di catturare la massima massa possibile degli asteroidi, riducendo al minimo il consumo di propellente e, di conseguenza, il ΔV, per ottimizzare la traiettoria.

La *reward function*, fondamentale nel processo di Decisione di Markov (*Markov Decision Process* - MDP), è progettata per fornire all'agente un feedback quantitativo sulla bontà delle sue azioni. Essa è definita in funzione della massa degli asteroidi catturati (normalizzata) e del ΔV speso per raggiungere gli asteroidi. Il reward per ogni azione è calcolata come la differenza tra il valore della *reward function* allo step $t+1$ e quello allo step t. Questo approccio consente di ottenere un valore negativo in caso di azioni inefficaci o controproducenti da parte dell'agente.

$$J = \frac{M_{ast\_normalizzata}}{\left(1 + \frac{\Delta V^{tot}}{50}\right)^2} \qquad \text{Reward Function} \tag{7.3}$$

Considerando che il moto degli asteroidi è assunto come Kepleriano, le dinamiche del problema sono completamente determinate e prevedibili. Di conseguenza, il fattore di *discount* utilizzato nel MDP è unitario ($\gamma = 1$), implicando che il valore del return per un episodio è calcolato come la somma dei reward ottenuti in ogni step dell'episodio. Questa impostazione enfatizza l'importanza di ogni singola azione intrapresa dall'agente durante la missione, poiché tutte le azioni contribuiscono direttamente al risultato finale.

$$J_{new} = J - J_{old} \qquad \text{Reward}$$

Politecnico di Torino

7.3 MCTS Implementation

7.3.1 Vanilla MCTS

L'algoritmo Monte Carlo Tree Search base, qui denominato MCTS *vanilla*, è implementato per interagire iterativamente con l'ambiente seguendo le quattro fasi distintive di questo metodo:

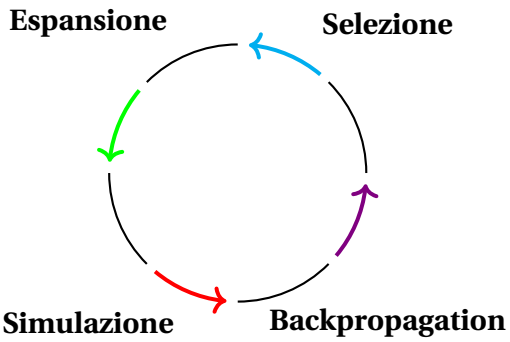


Figure 7.4: Schematizzazione delle fasi dell'algoritmo Vanilla MCTS.

Queste fasi consentono all'algoritmo di esplorare l'ambiente di simulazione in modo efficiente e di effettuare scelte informate basate sui risultati delle simulazioni precedenti.

Inizializzazione il Root-Node

Nel momento in cui inizia la simulazione, l'azione primaria consiste nel stabilire il *root node*, che si traduce nella creazione di un duplicato dell'ambiente reale. In seguito, l'algoritmo procede conducendo le sue operazioni sulla copia, lasciando così l'ambiente reale in uno stato non alterato.

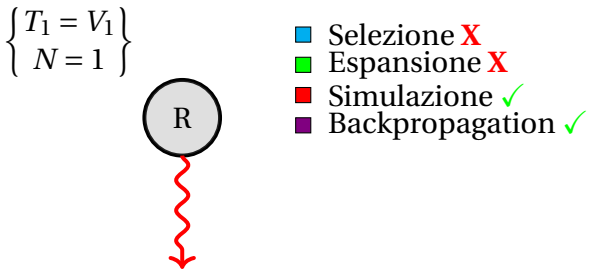


Figure 7.5: MCTS vanilla: Inizzializzazione del *root node*. Esempio pratico considerando unicamente 2 possibili nodi figli.

Inizialmente, in mancanza di un *tree search* preesistente, l'algoritmo procede con un roll-out, seguendo una policy di scelta casuale. Questa strategia prosegue fino al completamento dell'episodio simulato, con lo scopo principale di stimare il valore del *root node*, rappresentato da V_1. L'algoritmo, inoltre, memorizza attentamente questo valore stimato, oltre al numero di visite effettuate per ogni nodo.

Espansione il Root-Node

Una volta calcolato il valore approssimato del *root node*, avviene la fase di espansione per estendere il tree search. In questa fase, l'algoritmo prevede l'espansione di tutte le possibili azioni, determinate dalla maschera adottata nella simulazione. Per ogni azione possibile, viene generata una copia dell'ambiente, su cui si esegue uno step successivo.

Politecnico di Torino

Successivamente, si attribuisce ai *leaf node* appena aggiunti all'albero un valore dell' Upper Confidence Bound (UCB), impostato, in questa fase, come infinito.

$$UCT = \frac{T}{N} + c \cdot \sqrt{\frac{\ln(N_{parent})}{N_{children}}} \qquad \text{UCB\_score} \tag{7.4}$$

I figli del *root node* sono generati attraverso la creazione di una copia dell'ambiente.

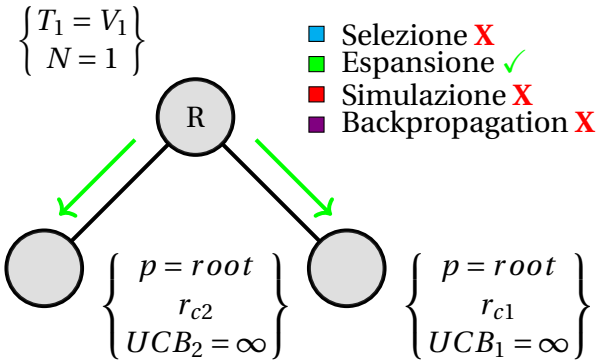
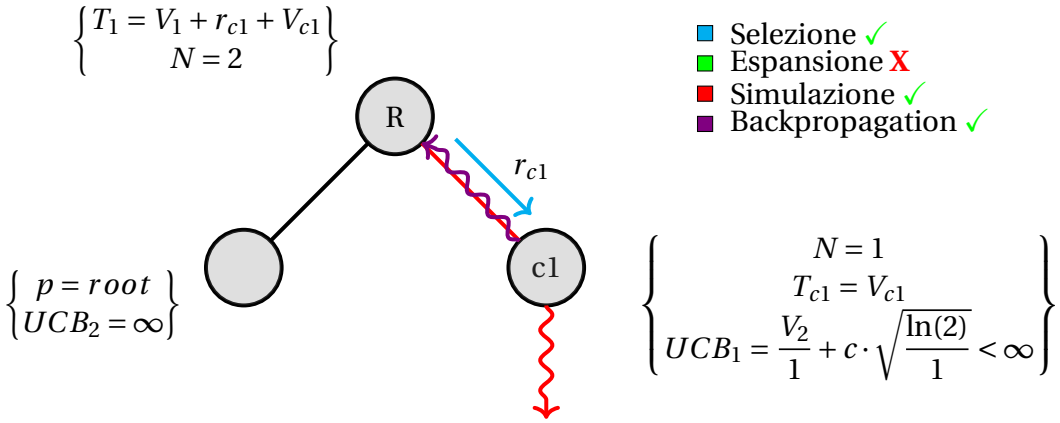


Figure 7.6: MCTS vanilla: fase di Espansione. Esempio pratico considerando unicamente 2 possibili nodi figli.

AGGIORNAMENTO DELLO SCORE UCB

Avendo definito l'Upper Confidence Bound (UCB) dei nodi espansi come infinito, l'algoritmo attua una scelta dell'azione da intraprendere in maniera casuale, ottenendo un reward correlato all'azione compiuta.

Segue un roll-out casuale, volto ad approssimare il valore del nodo espanso in esame. Una volta ottenuta questa approssimazione, si inizia la fase di backpropagation, in cui si aggiorna il valore di T_1 prendendo in considerazione il reward ottenuto ed il valore del *leaf node* ottenuto dal roll-out.



Dopo il roll-out, lo score UCB relativo al *children node* viene aggiornato a un valore reale, portando a:

$$UCB_2 > UCB_1$$

Di conseguenza, nella successiva iterazione, l'algoritmo scenderà lungo l'albero verso il *children node* che presenta ancora un UCB definito come infinito, ripetendo il processo di roll-out e backpropagation.

Politecnico di Torino

Questo ciclo iterativo, condotto dall'algoritmo un numero prefissato di volte, conduce allo sviluppo di un albero di ricerca (*search tree*) sempre più esteso. Man mano che la simulazione procede, la stima del valore dei nodi tende a una convergenza con il loro valore medio.

$$UCT = \underbrace{\frac{T}{N}}_{Exploitation} + \underbrace{c \cdot \sqrt{\frac{\ln(N_{parent})}{N_{children}}}}_{Exploration}$$

Inoltre il valore dell'*Upper Confidence Bound* (UCB) con l'aumentare della conoscenza dell'ambiente da parte dell'algoritmo, modifica il peso che attribuisce all'esplorazione. Infatti l'UCB è diviso in due parti: una incentrata sull' exploitation e l'altra sull'exploration. Con l'accumularsi delle iterazioni, la componente di exploration nell'UCB si riduce, poiché l'algoritmo guadagna fiducia nel valore effettivo dei nodi e, di conseguenza, tende a esplorare meno, orientandosi verso una strategia più *greedy*.

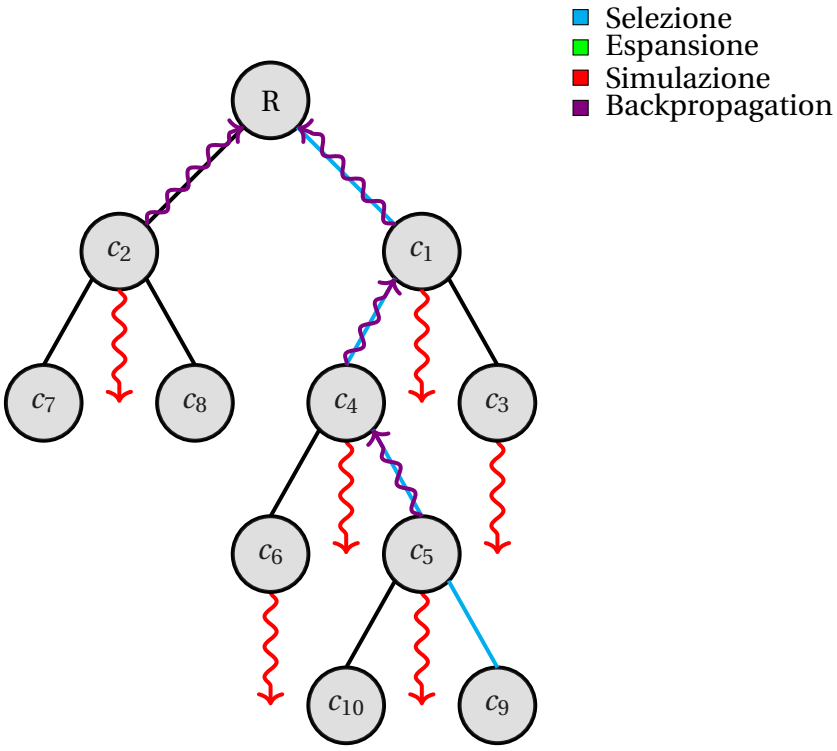


Figure 7.7: Search Tree - MCTS vanilla

Al termine del numero di iterazioni prefissato, l'agente procede all'azione nell'ambiente reale[4]. La scelta dell'azione da intraprendere si basa sul criterio di selezione che privilegia l'azione che è stata visitata il maggior numero di volte durante la fase di *planning*. In altre parole, l'azione selezionata corrisponde a quella associata al numero di visite più elevato, ovvero il massimo valore di N.

Una volta che l'azione è stata eseguita, vi è una modifica nel *root node*, il quale diventa il *child node* corrispondente all'azione appena intrapresa. A questo punto, il tree search viene ridimensionato, mantenendo in memoria solo la porzione che segue il nuovo *root node*. Il processo quindi si rinnova, seguendo questo ciclo di iterazioni fino al completamento dell'episodio.

[4]Ricordiamo che infatti l'algoritmo MCTS appartiene alla classe dei *decision-time planning*

Politecnico di Torino

7.3.2 AlphaZero

AlphaZero, sviluppato da DeepMind, rappresenta una svolta nel campo dell'apprendimento automatico e dei giochi strategici. Questo algoritmo segna un progresso significativo rispetto al tradizionale Monte Carlo Tree Search (MCTS) utilizzato in implementazioni precedenti come AlphaGo. Diversamente da questi ultimi, che si basavano su tecniche di ricerca sofisticate, adattamenti specifici al dominio e funzioni di valutazione ad hoc raffinate da esperti umani per decenni, AlphaZero combina tecniche di deep learning con MCTS.

In particolare, AlphaZero utilizza un approccio di reinforcement learning da tabula rasa, iniziando da una riproduzione casuale e senza alcuna conoscenza preimpostata dell'ambiente, tranne che per le leggi che regolano il problema. Questo permette all'algoritmo di apprendere autonomamente strategie di gioco efficaci in giochi complessi come Go, scacchi e anche la missione di nostro interesse.

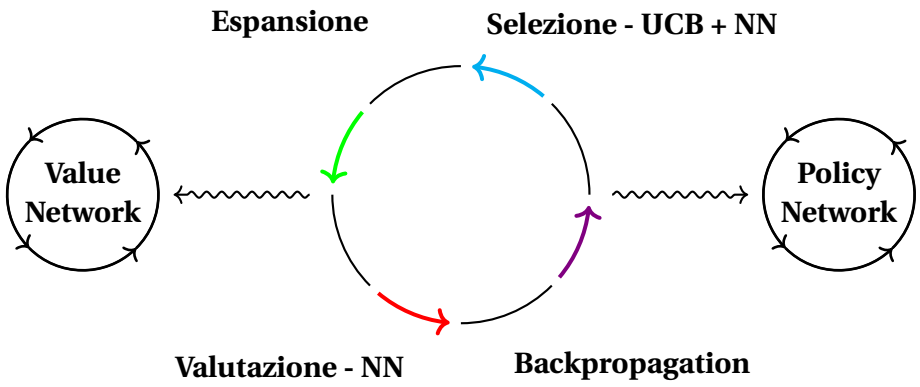


Figure 7.8: Schematizzazione delle fasi dell'algoritmo Alphazero.

Nel suo utilizzo del MCTS, AlphaZero implementa un processo iterativo e migliorato che si sviluppa attraverso le quattro fasi chiave:

1. **Selezione:** Innovando rispetto al classico MCTS, AlphaZero impiega un modello di rete neurale per la selezione dei nodi. Questa rete, rappresentata dalla funzione $f_\theta(s)$, analizza lo stato corrente del gioco s e produce un vettore di probabilità di mossa p, dove ogni componente $p_a = \Pr(a|s)$ corrisponde alla probabilità di ogni possibile azione a.

2. **Espansione:** AlphaZero può selezionare per l'espansione solamente i nodi che, secondo le valutazioni delle sue reti neurali, risultano essere i più promettenti. Questo metodo focalizzato contrasta con l'approccio di espansione più generico e meno mirato del MCTS vanilla, permettendo ad AlphaZero di concentrare la sua attenzione sulle mosse più strategiche.

   Tuttavia, nell'adattamento di AlphaZero al problema specifico in questione, questa caratteristica distintiva non è stata implementata. Si è optato per continuare ad adottare un'espansione uniforme, considerando tutte le azioni possibili date dalla maschera di gioco.

Politecnico di Torino

3. **Valutazione:** Una deviazione radicale dal MCTS vanilla si verifica nella fase di simulazione. Invece di eseguire simulazioni di partite complete a partire dai nodi espansi, AlphaZero elimina totalmente questa fase, affidandosi completamente alle valutazioni fornite dalla sua rete neurale per analizzare i nodi. Infatti, un secondo modello di rete neurale calcola un valore scalare v che stima l'*expected return* z dalla posizione corrente s, con $v \approx \mathbb{E}[z|s]$.

4. **Backpropagation:** Nella fase di backpropagation, AlphaZero utilizza i risultati ottenuti dai nodi esplorati per affinare le stime di valore lungo il percorso di ritorno alla radice dell'albero. Questa metodologia, che combina i risultati del gioco con le valutazioni della rete neurale, migliora notevolmente l'accuratezza del processo decisionale rispetto al tradizionale MCTS vanilla.

AlphaZero adotta un'approccio rivoluzionario nell'impiego di due Deep Neural networks per la valutazione delle azioni (*Policy Network*) e del valore degli stati (*Value Network*). Queste DNN sono meticolosamente addestrate attraverso un metodo di reinforcement learning basato su *self-play*, partendo da un insieme di parametri iniziali θ generati casualmente.

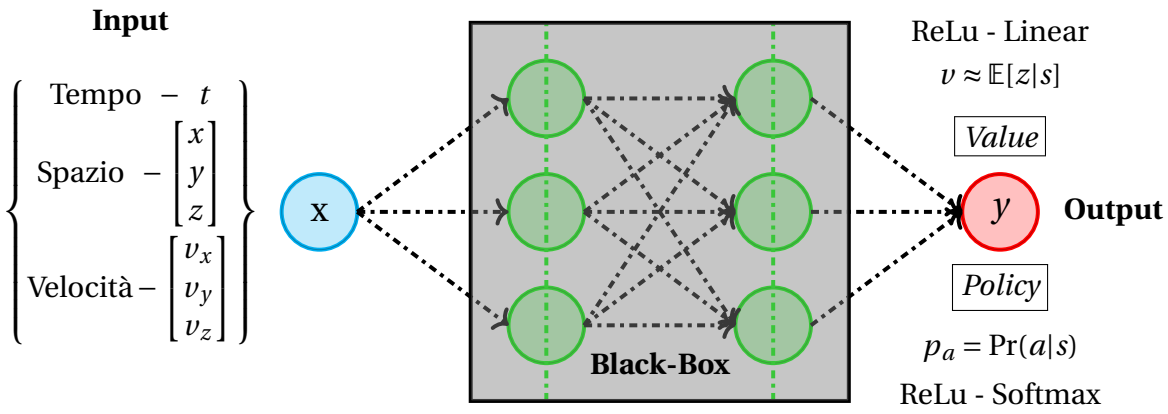


Figure 7.9: DNN in AlphaZero. In questa rappresentazione gli *hidden layer* della rete sono visti come una black-box in quanto il numero degli iperparametri (ovvero numero di neuroni e layer) è un parametro da scegliere accuratamente in fase di simulazione.

1. **Policy Network:** La *policy network* di AlphaZero è progettata per predire la probabilità di ciascuna possibile azione da un generico stato. Questa componente della rete analizza la configurazione corrente dell'ambiente e genera un vettore di probabilità che riflette la fattibilità e la potenziale efficacia di ogni azione possibile.

   A differenza del MCTS vanilla, dove la selezione delle mosse è guidata da un processo di ricerca che può essere più casuale e meno specifico, la rete di politica di AlphaZero fornisce un metodo di selezione basato su un'analisi approfondita e accurata.

2. **Value Network:** Parallelamente, la *Value Network* stima il valore strategico dello stato attuale dell'ambiente. Utilizzando una serie di caratteristiche estratte dalla configurazione dello stato, questa componente della rete valuta il return ricavabile partendo dallo stato attuale, aiutando a guidare la ricerca dell'algoritmo verso azioni e strategie che aumentano la probabilità di aumentare il return.

   La capacità di stimare accuratamente il valore di uno stato permette ad AlphaZero di effettuare scelte informate, anche in scenari di gioco complessi e variabili.

Durante ogni simulazione, lo stato finale viene esaminato attentamente secondo le regole specifiche del problema per determinare il risultato finale o il return z. Attraverso le informazioni ottenute nell'ambiente reale è possibile allenare i parametri della rete neurale θ, aggiornandoli ed affinandoli in modo iterativo lungo gli episodi. Questo aggiornamento si propone di ridurre la discrepanza tra il valore previsto dalla rete v_t e il risultato effettivo del gioco z, oltre a incrementare la corrispondenza tra il vettore di politica p_t e le probabilità di ricerca π_t generate dal MCTS.

Il processo di aggiornamento dei parametri θ avviene attraverso un metodo di *gradient descent* su una funzione di perdita l, che combina gli errori quadratici medi con la *cross entropy loss*[5]. Questa sofisticata tecnica di ottimizzazione consente alla rete neurale di AlphaZero di apprendere e di adattarsi in modo efficiente, migliorando progressivamente la sua capacità di valutare posizioni e di selezionare mosse strategiche. Il risultato di questo processo di apprendimento continuo è una rete neurale estremamente potente e versatile, capace di prendere decisioni strategiche complesse e di affrontare le sfide poste da giochi di strategia di alta complessità.

Inizializzazione ed Espansione del Root-Node

Nel momento in cui inizia la simulazione, l'algoritmo interroga la *value network* che restituisce il valore del *root node*. Successivamente, in funzione delle azioni disponibili fornite dalla maschera, avviene l'enspansione dei nodi figli seguita nuovamente dall'interrogazione della *value network* per ottenere il loro valore approssimato.

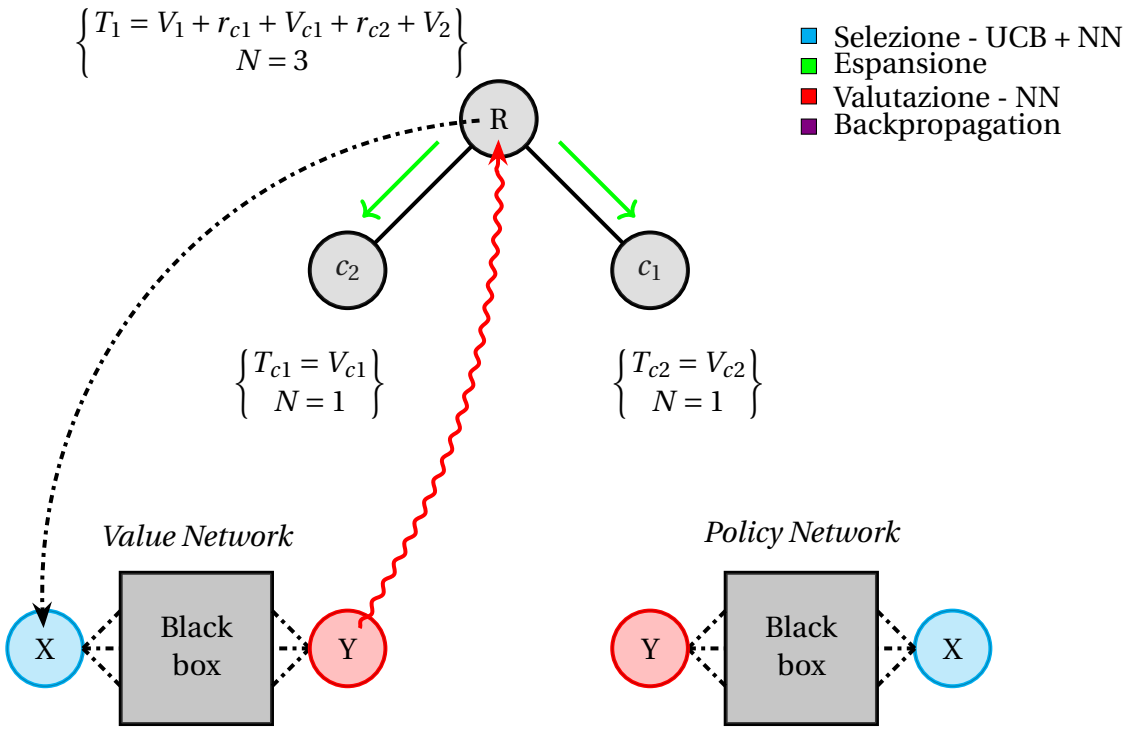


Figure 7.10: AlphaZero: Inizzializzazione ed Espansione del *root-node*. Per eseguire la valutazione del *root-node* l'algoritmo sfrutta la predizione della *Value Network*.

[5]Una funzione matematica utilizzata durante il processo di apprendimento per misurare la discrepanza tra le probabilità previste dalla rete neurale e le effettive distribuzioni delle probabilità delle azioni prese durante il gioco. In termini semplici, questa funzione aiuta a valutare quanto bene la rete neurale sia in grado di prevedere le mosse corrette in un gioco.

Politecnico di Torino

SELEZIONE

Nella fase di selezione, per scendere lungo il search tree, AlphaZero interroga la *policy network*, che generare un vettore di probabilità di azione p. Dove ogni componente di questo vettore, $p_a = \Pr(a|s)$, rappresenta la probabilità di intraprendere una specifica azione a.

Nella selezione delle azioni, AlphaZero cerca un equilibrio tra il valore stimato v dello stato, basato sulle esperienze passate (UCB), e la probabilità di selezione p, dando priorità alle azioni considerate più promettenti dalla rete neurale.

$$PUCT = \frac{T}{N} + P_a(s,a) \cdot c \cdot \sqrt{\frac{\ln(N_{parent})}{N_{children}}} \qquad \text{PUTC} \tag{7.5}$$

Questa formula permette ad AlphaZero di combinare la ricerca basata su MCTS con intuizioni provenienti dalla rete neurale, guidando la selezione delle azioni in modo più informato e sofisticato rispetto al MCTS vanilla.

Il metodo tradizionale UCT viene così arricchito dall'output della rete neurale, rendendo il processo decisionale di AlphaZero non solo più accurato ma anche in grado di adattarsi e migliorare continuamente attraverso l'esperienza di gioco.

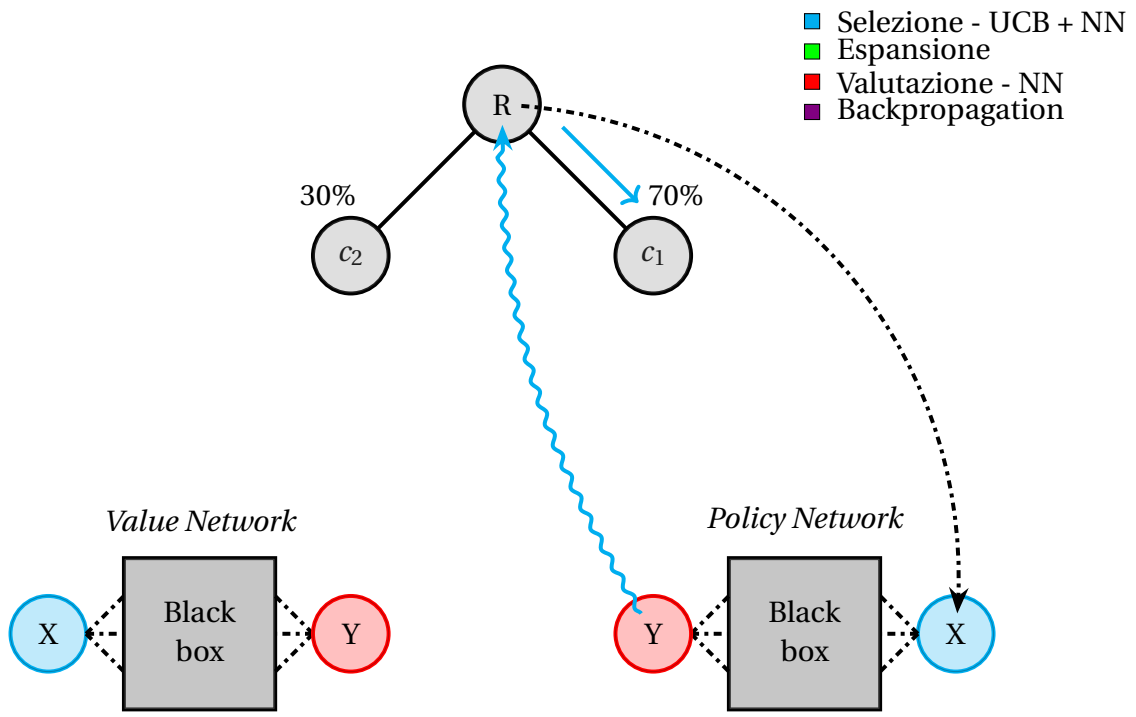


Figure 7.11: AlphaZero: fase di Selezione. L'algoritmo sfrutta la predizione della *Policy Network*, la quale restituisce la predizione della distribuzione di probabilità delle azioni. Per semplicità la rappresentazione considera unicamente 2 azioni possibili: c_1 (70%) e c_2 (30%).

Politecnico di Torino

ESPANSIONE & SIMULAZIONE

Dopo aver raggiunto un *leaf node* basandosi sulle previsioni della rete neurale, AlphaZero procede con l'aggiunta dei *child nodes*, che rappresentano azioni potenziali da esplorare. Questa fase viene svolta esattamente come nella precedente implementazione di MCTS, ovvero espandendo il *leaf node* considerando ogni possibile azione restituita dalla maschera in quel particolare stato.

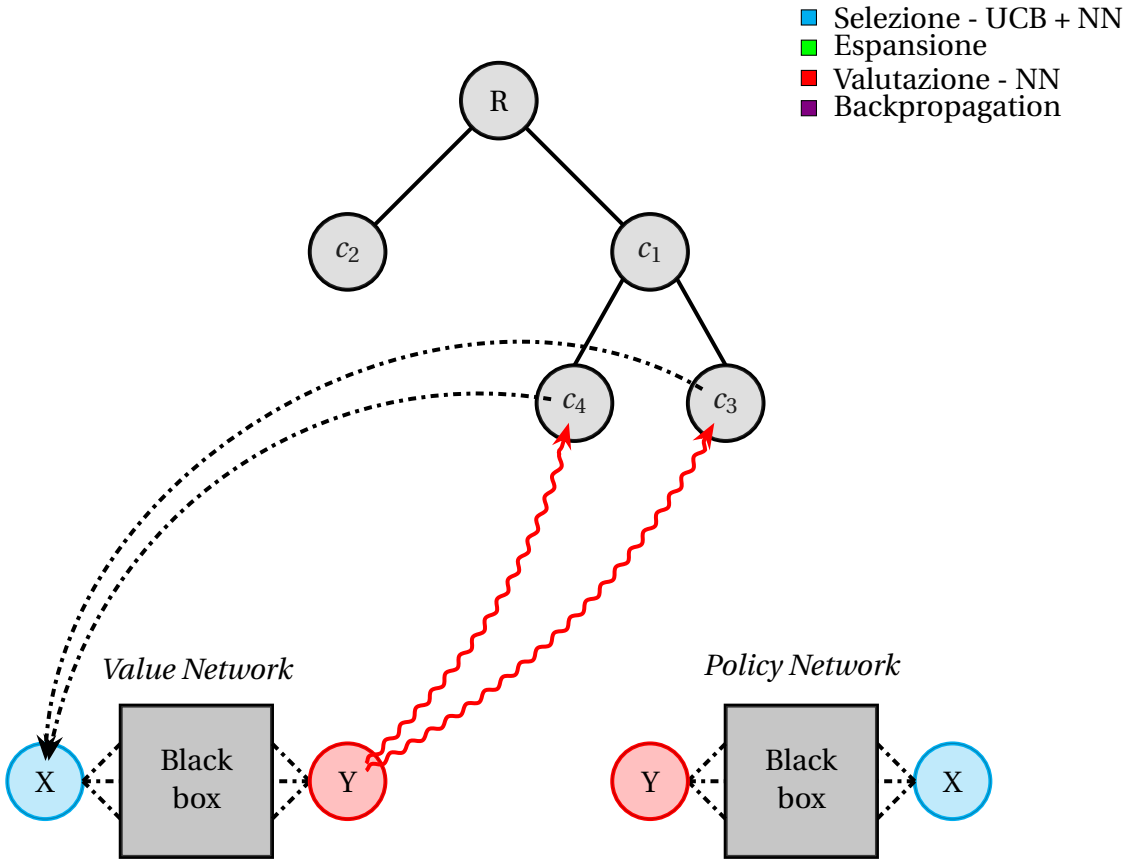


Figure 7.12: AlphaZero: fase di Espansione e Simulazione.

La fase di simulazione in AlphaZero rappresenta una deviazione sostanziale dalla metodologia tradizionale utilizzata nel MCTS vanilla. Invece di eseguire simulazioni complete a partire dai nodi espansi (roll-out), AlphaZero si avvale ancora del suo modello di deep learning per eseguire valutazioni dirette degli stati.

Questo approccio innovativo elimina la necessità di eseguire sequenze di azione complete fino alla loro conclusione, risultando in una drastica riduzione sia del tempo di calcolo che della complessità operativa. Inoltre, diversamente dall'approccio del MCTS vanilla basato su una *roll-out policy* casuale, l'algoritmo AlphaZero introduce un meccanismo di apprendimento e di aggiornamento continuativo della policy, tendendo a indicare un valore che si avvicina maggiormente all'ottimale piuttosto che a un valore medio.

Evitando simulazioni lunghe e dettagliate e concentrandosi su valutazioni rapide basate sull'apprendimento acquisito, AlphaZero adotta un metodo di risoluzione dei problemi che emula in qualche modo il processo decisionale umano, offrendo un esempio di come l'intelligenza artificiale possa integrare tecniche di problem-solving umane per ottimizzare le proprie prestazioni in contesti complessi.

Politecnico di Torino

BACKPROPAGATION

Nel processo di backpropagation, i risultati ottenuti dai nodi visitati vengono sfruttati per aggiornare le stime delle state value lungo il percorso di ritorno al *root node* dell'albero. Questo processo è cruciale per affinare la strategia di gioco dell'algoritmo.

Formalmente, la backpropagation in AlphaZero è rappresentata da un aggiornamento delle stime di valore dei nodi visitati, che riflettono una combinazione ponderata del valore stimato v, calcolato dalla rete neurale, e del risultato del gioco z. Questo processo di aggiornamento comporta la modifica dei pesi della rete neurale, affinando la capacità della rete di prevedere il valore v in modo sempre più accurato con l'accumularsi delle esperienze di gioco.

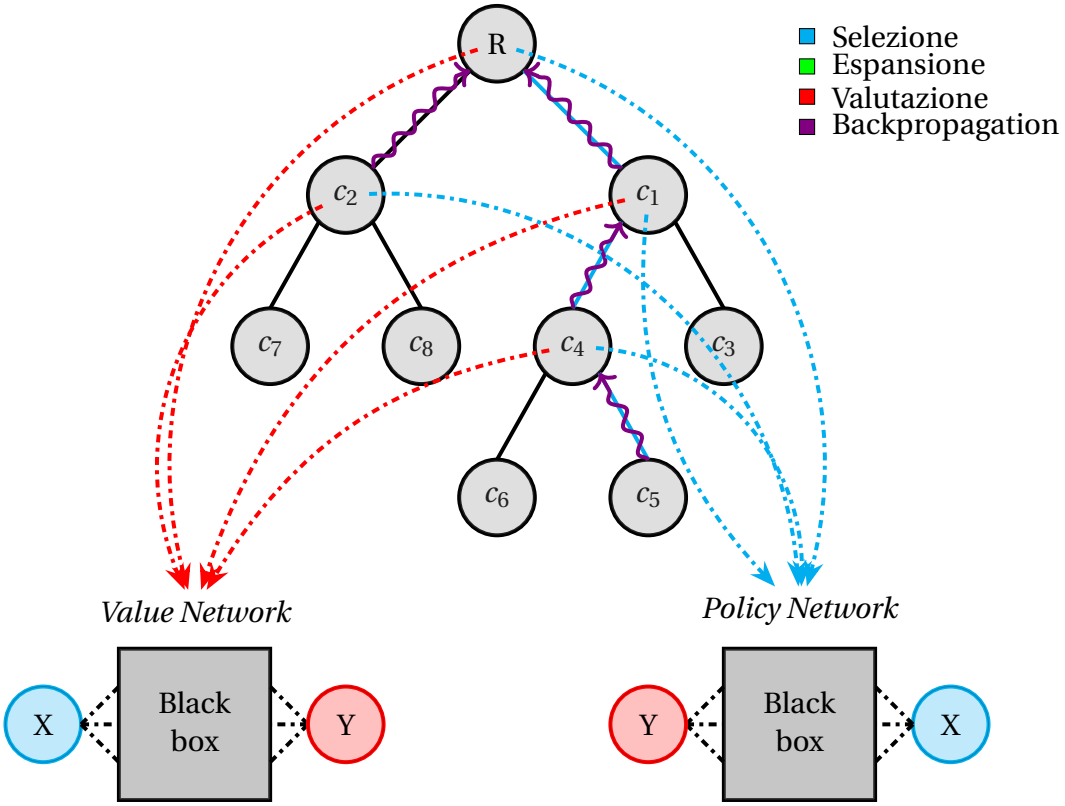


Figure 7.13: Search Tree - AlphaZero.

Questa metodologia presenta vantaggi significativi. Innanzitutto, permette di sfruttare la potenza delle reti neurali per ottenere valutazioni di stato profonde e strategiche, non limitate dalla lunghezza delle simulazioni di gioco. Inoltre, incorporando i risultati effettivi, la rete neurale si adatta e apprende da esperienze reali, migliorando costantemente nella sua precisione di valutazione.

Alla conclusione dell'episodio di gioco, si procede con l'allenamento delle reti neurali, utilizzando i *reward* ottenuti e il numero di azioni intraprese. Questo permette di affinare la *value network* e la *policy network*:

- **Value Network**: Affinamento basato sui *reward*, valutando l'accuratezza della predizione della rete.
- **Policy network**: Allenamento basato sulla percentuale di azioni intraprese, utilizzando la cross entropy loss.

CHAPTER 8

RISULTATI E CONCLUSIONI

REFERENCE SOLUTION

Le prestazioni ottenute dal team del Politecnico di Torino durante la competizione si pongono come un punto di riferimento per l'analisi delle soluzioni ottenute durante questo studio. Un *Performance Index* (J) medio di 14.86 stabilisce un criterio di valutazione elevato, fungendo da standard di confronto per gli approcci successivamente analizzati.

MotherShips	**J**	ΔV - [Km/s]	**Mass** - [kg]
#1	15.26966	30.257	$39.342 \cdot 10^{14}$
#2	16.41901	29.240	$41.238 \cdot 10^{14}$
#3	13.96417	31.581	$37.175 \cdot 10^{14}$
#4	14.69212	32.698	$40.191 \cdot 10^{14}$
#5	14.61272	31.125	$38.468 \cdot 10^{14}$
#6	14.06594	33.048	$38.804 \cdot 10^{14}$
#7	14.90281	31.099	$39.208 \cdot 10^{14}$
#8	14.60292	31.600	$38.894 \cdot 10^{14}$
#9	15.00475	31.792	$40.152 \cdot 10^{14}$
#10	15.19233	32.922	$41.785 \cdot 10^{14}$
	14.86	315.36	$39.5258 \cdot 10^{15}$

Table 8.1: Reference Solution

I risultati ottenuti dal team fungono da standard di qualità di riferimento, evidenziando l'efficacia e l'efficienza delle strategie messe in atto. Questo offre un benchmark di valore per gli algoritmi MCTS Vanilla e AlphaZero esaminati in questa analisi.

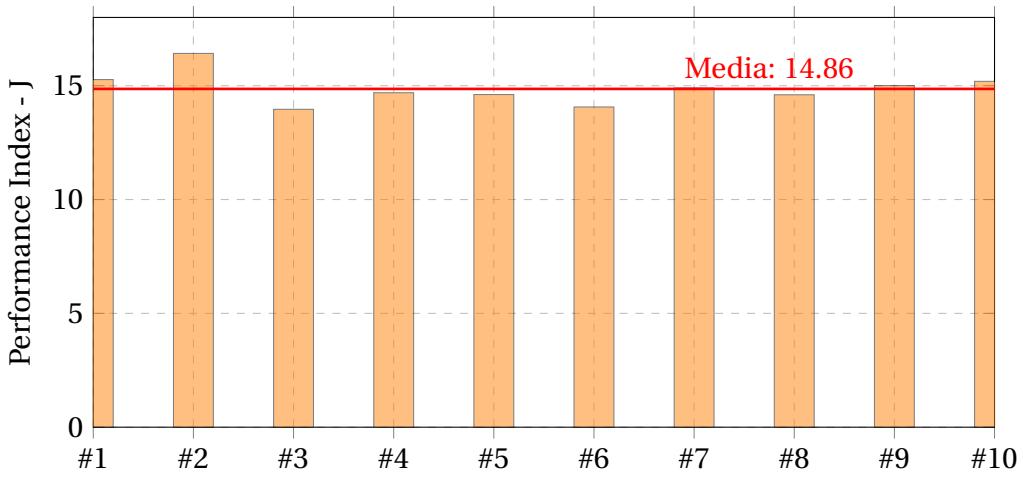
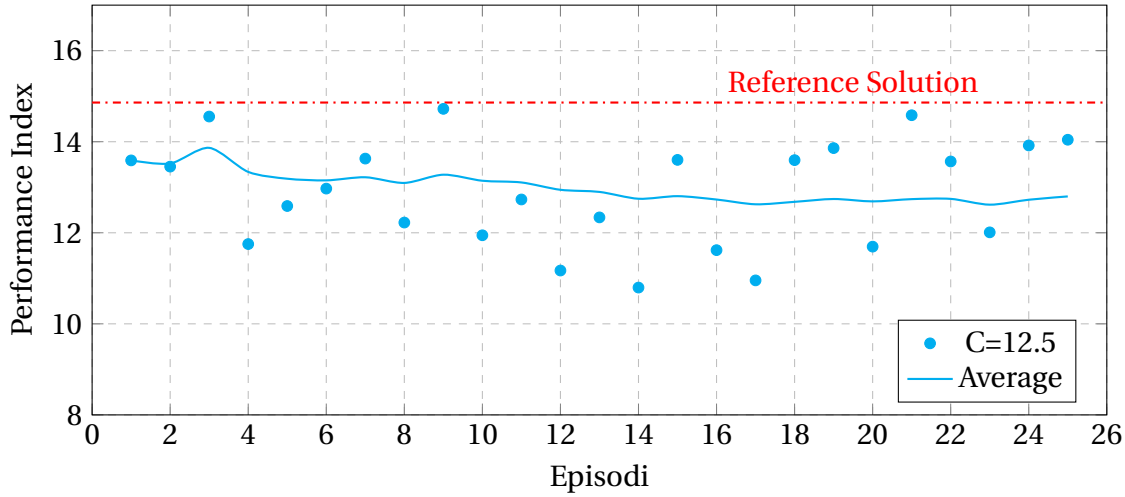


Table 8.2: Performance Index medio delle reference solution.

L'analisi comparativa che ne deriva trascende il solo confronto numerico, ma getta luce sulle capacità e sui limiti intrinsechi dei metodi esplorati per affrontare e risolvere la problematica oggetto di studio.

RISULTATI - MCTS VANILLA

I risultati derivati dalle simulazioni effettuate mediante l'implementazione dell'algoritmo MCTS Vanilla evidenziano un marcato divario rispetto al valore della *Reference Solution*.



$$\begin{cases} C_{UCB} = 12.5 \\ \text{Exploring Constant} = 4500 \\ \text{Time} = 4-20:00:13 \\ \text{Memory} = 37.93\ GB \end{cases}$$

Risultati Ottenuti	
Minimo	10.7965
Massimo	14.7217
Media	12.79868

Figure 8.1: Risultati MCTS Vanilla

Il valore della soluzione ottenuta è influenzata significativamente da due fattori principali:

1. **Costante UCB** (C_{UCB}): Questa costante determina l'equilibrio tra esplorazione ed esploitation all'interno della *Tree Policy*, influenzando direttamente la capacità dell'algoritmo di scoprire e valutare nuove azioni rispetto all'approfondimento di quelle già esplorate.
2. **Exploring Constant**: Tale parametro definisce la profondità di esplorazione dell'albero decisionale prima di compiere un'azione nell'ambiente reale, agendo come un indicatore di quanto l'algoritmo pensa ed analizza le possibili mosse prima di decidere.

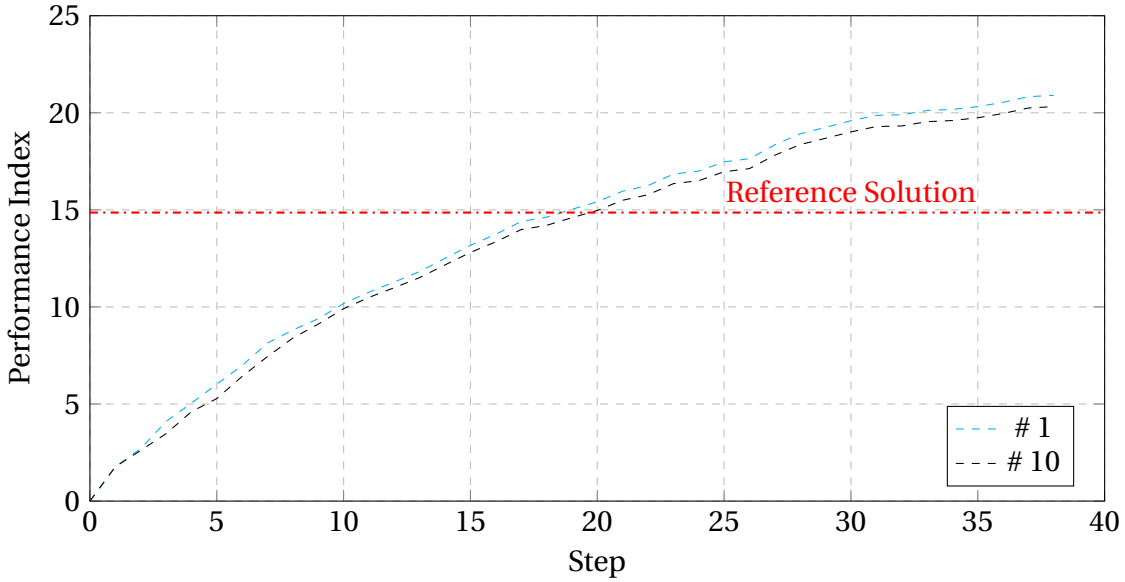
Un'alta configurazione della C_{UCB} stimola l'algoritmo a privilegiare l'esplorazione di nuove azioni, potenzialmente scoprendo sequenze di azioni più efficaci in termini di *expected return* rispetto a quelle già note. Tale dinamica è fondamentale per evitare ottimi locali, spingendo l'algoritmo a esplorare soluzioni che, a prima vista, potrebbero sembrare meno promettenti.

Allo stesso modo, incrementare l'*Exploring Constant* amplifica l'abilità dell'algoritmo di esaminare un maggior numero di combinazioni stato-azione, estendendo la fase di *planning* e fornendo all'algoritmo una base di conoscenza più ampia per la selezione dell'azione ottimale. Questo processo, tuttavia, comporta un inevitabile aumento dei tempi e dei costi computazionali associati alla simulazione, che deve rimanere entro i limiti temporali definiti dall'infrastruttura di calcolo a disposizione[1].

[1] In particolare, i cluster del Politecnico di Torino, quali Legion ed Hactar, impongono un limite massimo di 5 giorni per il completamento delle simulazioni.

RISULTATI - ALPHAZERO

L'adozione dell'approccio AlphaZero ha rappresentato una svolta nel panorama degli algoritmi di ottimizzazione, manifestando un avanzamento notevole in termini di Performance Index. Tale progresso dimostra che la fusione delle tecniche di *Deep Learning* (DL) con il framework Monte Carlo Tree Search (MCTS) migliora notevolmente la qualità delle soluzioni ricavate.



- $C_{PUCB} = 1.25$
- Exploring Constant = 250
- Episodi = 3000
- NN = [256] × 3
- Time = 4 – 07 : 59 : 10
- Memory = 18.66 *GB*

Risultati Ottenuti	
Miglior Episodio # 1	20.899
Miglior Episodio # 10	20.319
Media	16.42

Figure 8.2: Migliori Episodi: Alphazero

Le prestazioni e l'efficacia delle soluzioni ottenute sono significativamente influenzate da diversi fattori chiave:

1. **Costante PUCB** (C_{PUCB}): Analogamente all'implementazione precedente, questo parametro regola il bilanciamento tra esplorazione ed esploitation all'interno della *Tree Policy*. Tuttavia, nell'algoritmo AlphaZero, la C_{PUCB} è arricchita dalla moltiplicazione per la probabilità a priori derivante dalla *Policy Network*.

   Inoltre, in questa implementazione, all'interno della *Tree Policy* avviene una normalizzazione del valore T/N, comportando una variazione significativa dell'ordine di grandezza della C_{PUCB}.

2. **Exploring Constant**: Riafferma l'importanza della profondità di esplorazione dell'albero decisionale prima che l'algoritmo proceda con un'azione concreta, mantenendo il suo ruolo cruciale nell'indicare la profondità della ricerca.

Politecnico di Torino

3. **Episodi**: Il numero di episodi giocati durante la simulazione acquista una nuova rilevanza in AlphaZero, diventando un fattore determinante per l'efficacia dell'allenamento delle reti neurali. Un numero di episodi inappropriato può condurre a una soluzione non ottimale, caratterizzata da un *underfitting* (se troppo basso), o dall'*overfitting* delle reti neurali (se eccessivamente elevato), influenzando direttamente la capacità dell'algoritmo.

4. **Architettura delle Reti Neurali**: La configurazione strutturale delle reti neurali riveste una notevole importanza in AlphaZero. La selezione delle architetture delle NN, inclusi il numero di layers, i neuroni per layer e le funzioni di attivazione, incide profondamente sulle prestazioni delle predizioni effettuate dall'algoritmo. Una configurazione non ottimale può risultare in previsioni che o trascurano l'essenza dei dati di allenamento o, al contrario, enfatizzano eccessivamente le anomalie, compromettendo la qualità delle soluzioni elaborate.

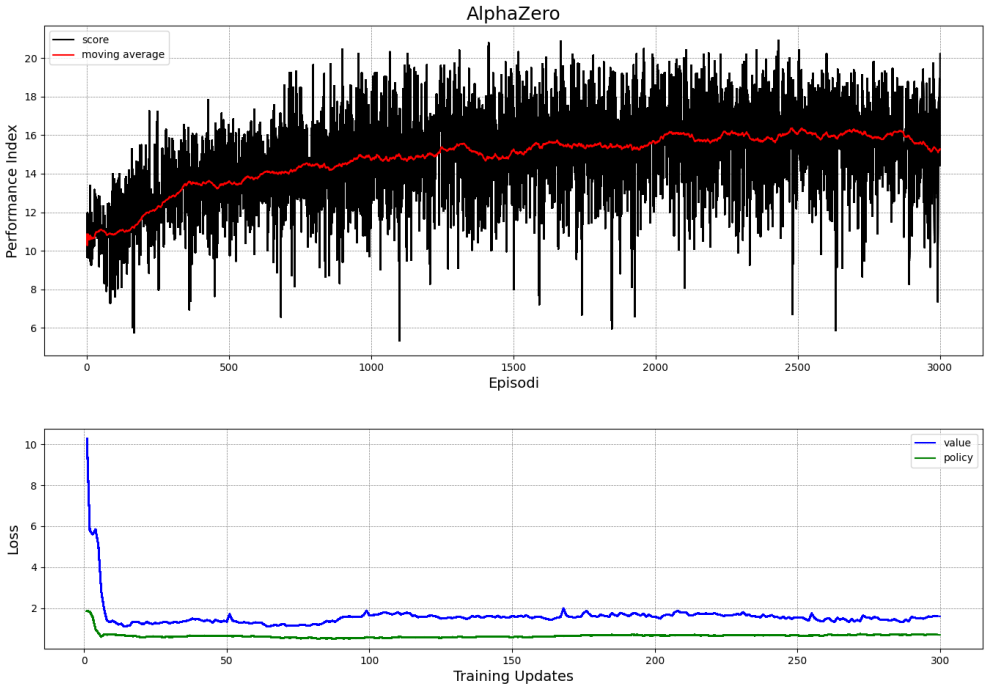


Figure 8.3: Andamento del *Performance Index* e delle funzioni di perdita durante la simulazione al variare degli episodi.

Con l'adozione di AlphaZero, l'accento si sposta significativamente verso l'ottimizzazione dell'addestramento delle reti neurali, in virtù della loro capacità di apprendere dinamicamente dall'ambiente e di generare valutazioni precise delle mosse future. Ridurre l'*Exploring Constant* consente quindi di allocare risorse computazionali più consistenti verso l'iterazione e il raffinamento delle reti neurali, elementi chiave per la costruzione di un modello predittivo efficiente.

L'obiettivo è raggiungere un equilibrio ottimale dove il tempo di *planning* è ridotto ma sufficientemente efficace grazie al supporto di predizioni neurali accurate, permettendo all'algoritmo di navigare con destrezza nello spazio delle possibili soluzioni. In questo contesto, la fase di *planning* non viene semplicemente accorciata, ma ricalibrata per integrarsi sinergicamente con l'allenamento derivante dalle reti neurali.

MJD	ID	MASS - [kg]	ΔV - [km/s]	MANEUVER	REWARD	TOTAL MASS - [kg]	TOTAL ΔV - [km/s]	SCORE
95739	-1	$0 \cdot 10^{14}$	7.07	single_impulse	0	$0 \cdot 10^{14}$	7.07	0
96069	82869	$1.841 \cdot 10^{14}$	4.599	single_impulse	1.765	$1.841 \cdot 10^{14}$	11.669	1.765
96279	57956	$1.504 \cdot 10^{14}$	0.697	single_impulse	0.934	$3.345 \cdot 10^{14}$	12.366	2.699
96369	72526	$1.865 \cdot 10^{14}$	1.351	single_impulse	1.401	$5.21 \cdot 10^{14}$	13.717	4.1
96519	56416	$1.508 \cdot 10^{14}$	0.609	single_impulse	0.942	$6.719 \cdot 10^{14}$	14.326	5.042
96669	45370	$1.475 \cdot 10^{14}$	1.653	single_impulse	0.979	$8.193 \cdot 10^{14}$	15.979	6.021
96819	81718	$1.854 \cdot 10^{14}$	0.57	single_impulse	0.961	$10.047 \cdot 10^{14}$	16.549	6.982
97029	75848	$1.895 \cdot 10^{14}$	1.244	single_impulse	1.161	$11.942 \cdot 10^{14}$	17.793	8.143
97179	34071	$1.512 \cdot 10^{14}$	1.523	single_impulse	0.665	$13.454 \cdot 10^{14}$	19.316	8.809
97269	75338	$1.607 \cdot 10^{14}$	1.239	single_impulse	0.583	$15.061 \cdot 10^{14}$	20.555	9.392
97419	60627	$1.898 \cdot 10^{14}$	1.509	single_impulse	0.782	$16.959 \cdot 10^{14}$	22.064	10.174
97569	38707	$1.824 \cdot 10^{14}$	0.879	single_impulse	0.585	$18.784 \cdot 10^{14}$	22.943	10.759
97659	78165	$1.443 \cdot 10^{14}$	1.477	single_impulse	0.524	$20.226 \cdot 10^{14}$	24.42	11.283
97869	82336	$1.917 \cdot 10^{14}$	0.78	single_impulse	0.542	$22.143 \cdot 10^{14}$	25.2	11.825
98019	47221	$1.812 \cdot 10^{14}$	0.595	single_impulse	0.681	$23.955 \cdot 10^{14}$	25.795	12.506
98109	50963	$1.716 \cdot 10^{14}$	0.468	single_impulse	0.668	$25.671 \cdot 10^{14}$	26.263	13.174
98319	72657	$1.466 \cdot 10^{14}$	0.77	single_impulse	0.568	$27.137 \cdot 10^{14}$	27.033	13.742
98409	64856	$1.926 \cdot 10^{14}$	1.293	single_impulse	0.658	$29.063 \cdot 10^{14}$	28.326	14.4
98559	73759	$1.531 \cdot 10^{14}$	0.816	single_impulse	0.222	$30.594 \cdot 10^{14}$	29.142	14.622
98769	78910	$1.567 \cdot 10^{14}$	0.951	single_impulse	0.408	$32.161 \cdot 10^{14}$	30.093	15.03
98919	34450	$1.689 \cdot 10^{14}$	0.803	single_impulse	0.386	$33.85 \cdot 10^{14}$	30.896	15.415
99009	50018	$1.961 \cdot 10^{14}$	1.136	single_impulse	0.545	$35.811 \cdot 10^{14}$	32.032	15.96
99219	50755	$1.766 \cdot 10^{14}$	0.566	single_impulse	0.29	$37.576 \cdot 10^{14}$	32.598	16.25
99369	55616	$1.951 \cdot 10^{14}$	1.318	single_impulse	0.592	$39.527 \cdot 10^{14}$	33.916	16.842
99519	52834	$1.748 \cdot 10^{14}$	0.665	single_impulse	0.155	$41.275 \cdot 10^{14}$	34.581	16.997
99609	55196	$1.883 \cdot 10^{14}$	1.124	single_impulse	0.476	$43.158 \cdot 10^{14}$	35.705	17.473
99699	40022	$1.653 \cdot 10^{14}$	0.092	single_impulse	0.161	$44.811 \cdot 10^{14}$	35.797	17.634
99909	40313	$1.927 \cdot 10^{14}$	0.417	single_impulse	0.716	$46.738 \cdot 10^{14}$	36.214	18.35
100119	63261	$1.949 \cdot 10^{14}$	0.707	single_impulse	0.567	$48.688 \cdot 10^{14}$	36.921	18.917
100269	50093	$1.737 \cdot 10^{14}$	0.605	single_impulse	0.334	$50.425 \cdot 10^{14}$	37.526	19.251
100359	29329	$1.658 \cdot 10^{14}$	0.9	single_impulse	0.339	$52.083 \cdot 10^{14}$	38.426	19.59
100569	47601	$1.912 \cdot 10^{14}$	1.084	single_impulse	0.278	$53.995 \cdot 10^{14}$	39.51	19.868
100719	35331	$1.508 \cdot 10^{14}$	0.988	single_impulse	0.028	$55.503 \cdot 10^{14}$	40.498	19.896
100929	52749	$1.955 \cdot 10^{14}$	1.207	single_impulse	0.222	$57.458 \cdot 10^{14}$	41.705	20.118
101079	22004	$1.817 \cdot 10^{14}$	1.072	single_impulse	0.056	$59.275 \cdot 10^{14}$	42.777	20.174
101169	66462	$1.92 \cdot 10^{14}$	0.698	single_impulse	0.142	$61.195 \cdot 10^{14}$	43.475	20.316
101319	56504	$1.674 \cdot 10^{14}$	0.612	single_impulse	0.224	$62.869 \cdot 10^{14}$	44.087	20.541
101529	81966	$1.777 \cdot 10^{14}$	1.037	single_impulse	0.288	$64.646 \cdot 10^{14}$	45.124	20.829
101739	78576	$1.756 \cdot 10^{14}$	0	coast	0.071	$66.401 \cdot 10^{14}$	45.124	20.899

Table 8.3: Migliore #1 episodio AlphaZero.

Politecnico di Torino

Data	06/06/2137
Performance Index	$J = 20.899$
Massa	$M = 66.40134 \cdot 10^{14}$
Asteroidi Visitati	38
Total ΔV	$\Delta V = 45.124$
Asteroid ID	78576

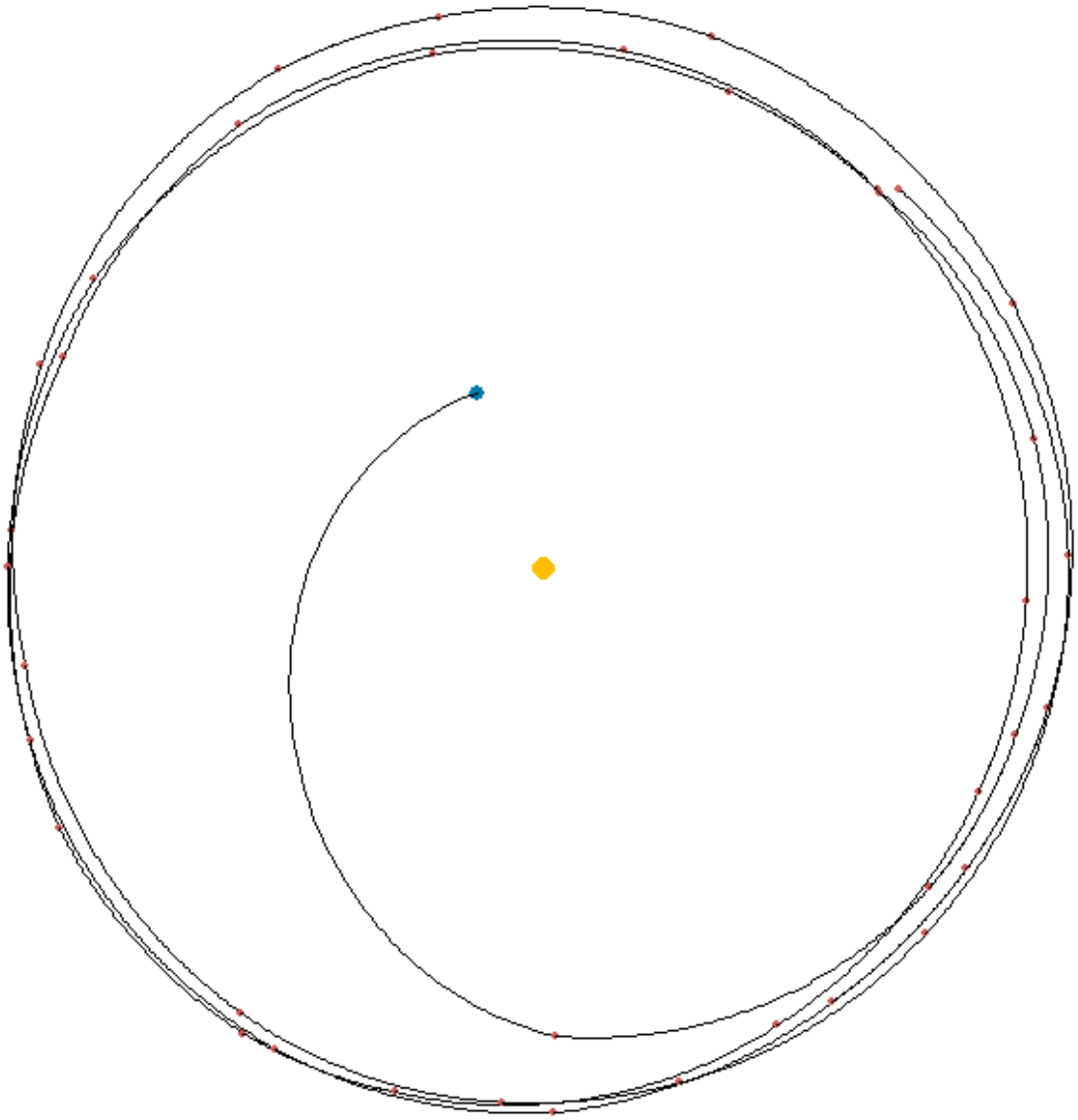


Figure 8.4: Rappresentazione delle traiettorie effettuate dalla *Mother Ships* durante l'episodio #1.

MJD	ID	MASS - [kg]	ΔV - [km/s]	MANEUVER	REWARD	TOTAL MASS - [kg]	TOTAL ΔV - [km/s]	SCORE
95739	-1	$0 \cdot 10^{14}$	7.07	single_impulse	0	$0 \cdot 10^{14}$	7.07	0
96069	82869	$1.841 \cdot 10^{14}$	5.02	single_impulse	1.765	$1.841 \cdot 10^{14}$	12.09	1.765
96219	51383	$1.441 \cdot 10^{14}$	2.632	single_impulse	0.843	$3.282 \cdot 10^{14}$	14.722	2.608
96309	57956	$1.504 \cdot 10^{14}$	1.38	single_impulse	0.862	$4.786 \cdot 10^{14}$	16.102	3.47
96399	72526	$1.865 \cdot 10^{14}$	2.055	single_impulse	1.133	$6.651 \cdot 10^{14}$	18.157	4.603
96489	56416	$1.508 \cdot 10^{14}$	0.533	single_impulse	0.676	$8.16 \cdot 10^{14}$	18.69	5.28
96639	50694	$1.935 \cdot 10^{14}$	0.658	single_impulse	1.142	$10.095 \cdot 10^{14}$	19.348	6.422
96849	81718	$1.854 \cdot 10^{14}$	0.847	single_impulse	1.022	$11.949 \cdot 10^{14}$	20.195	7.444
97059	75848	$1.895 \cdot 10^{14}$	0.9	single_impulse	0.954	$13.843 \cdot 10^{14}$	21.095	8.398
97209	75338	$1.607 \cdot 10^{14}$	1.068	single_impulse	0.718	$15.45 \cdot 10^{14}$	22.163	9.115
97419	60627	$1.898 \cdot 10^{14}$	1.429	single_impulse	0.792	$17.349 \cdot 10^{14}$	23.592	9.908
97569	38707	$1.824 \cdot 10^{14}$	0.879	single_impulse	0.584	$19.173 \cdot 10^{14}$	24.471	10.491
97659	78165	$1.443 \cdot 10^{14}$	1.477	single_impulse	0.502	$20.616 \cdot 10^{14}$	25.948	10.993
97869	82336	$1.917 \cdot 10^{14}$	0.78	single_impulse	0.52	$22.533 \cdot 10^{14}$	26.728	11.513
98019	47221	$1.812 \cdot 10^{14}$	0.595	single_impulse	0.653	$24.345 \cdot 10^{14}$	27.323	12.166
98109	50963	$1.716 \cdot 10^{14}$	0.468	single_impulse	0.641	$26.06 \cdot 10^{14}$	27.791	12.808
98319	72657	$1.466 \cdot 10^{14}$	0.77	single_impulse	0.545	$27.526 \cdot 10^{14}$	28.561	13.352
98409	64856	$1.926 \cdot 10^{14}$	1.293	single_impulse	0.633	$29.452 \cdot 10^{14}$	29.854	13.985
98559	73759	$1.531 \cdot 10^{14}$	0.816	single_impulse	0.216	$30.983 \cdot 10^{14}$	30.67	14.201
98769	78910	$1.567 \cdot 10^{14}$	0.951	single_impulse	0.394	$32.55 \cdot 10^{14}$	31.621	14.596
98919	34450	$1.689 \cdot 10^{14}$	0.803	single_impulse	0.373	$34.239 \cdot 10^{14}$	32.424	14.969
99009	50018	$1.961 \cdot 10^{14}$	1.136	single_impulse	0.527	$36.2 \cdot 10^{14}$	33.56	15.495
99219	50755	$1.766 \cdot 10^{14}$	0.566	single_impulse	0.283	$37.966 \cdot 10^{14}$	34.126	15.778
99369	55616	$1.951 \cdot 10^{14}$	1.318	single_impulse	0.571	$39.916 \cdot 10^{14}$	35.444	16.349
99519	52834	$1.748 \cdot 10^{14}$	0.665	single_impulse	0.154	$41.664 \cdot 10^{14}$	36.109	16.504
99609	55196	$1.883 \cdot 10^{14}$	1.124	single_impulse	0.461	$43.548 \cdot 10^{14}$	37.233	16.964
99699	40022	$1.653 \cdot 10^{14}$	0.092	single_impulse	0.16	$45.201 \cdot 10^{14}$	37.325	17.125
99909	40313	$1.927 \cdot 10^{14}$	0.417	single_impulse	0.69	$47.128 \cdot 10^{14}$	37.742	17.814
100119	63261	$1.949 \cdot 10^{14}$	0.707	single_impulse	0.548	$49.077 \cdot 10^{14}$	38.449	18.362
100269	50093	$1.737 \cdot 10^{14}$	0.605	single_impulse	0.325	$50.814 \cdot 10^{14}$	39.054	18.688
100359	29329	$1.658 \cdot 10^{14}$	0.9	single_impulse	0.33	$52.472 \cdot 10^{14}$	39.954	19.017
100569	47601	$1.912 \cdot 10^{14}$	1.084	single_impulse	0.272	$54.385 \cdot 10^{14}$	41.038	19.29
100719	35331	$1.508 \cdot 10^{14}$	0.988	single_impulse	0.033	$55.893 \cdot 10^{14}$	42.026	19.322
100929	52749	$1.955 \cdot 10^{14}$	1.207	single_impulse	0.219	$57.848 \cdot 10^{14}$	43.233	19.542
101079	22004	$1.817 \cdot 10^{14}$	1.072	single_impulse	0.06	$59.664 \cdot 10^{14}$	44.305	19.602
101169	66462	$1.92 \cdot 10^{14}$	0.698	single_impulse	0.143	$61.585 \cdot 10^{14}$	45.003	19.744
101319	56504	$1.674 \cdot 10^{14}$	0.612	single_impulse	0.22	$63.259 \cdot 10^{14}$	45.615	19.964
101529	81966	$1.777 \cdot 10^{14}$	1.037	single_impulse	0.281	$65.035 \cdot 10^{14}$	46.652	20.246
101739	78576	$1.756 \cdot 10^{14}$	0	coast	0.074	$66.791 \cdot 10^{14}$	46.652	20.319

Table 8.4: Migliore #10 episodio AlphaZero.

Politecnico di Torino

CONCLUSIONI

La presente tesi magistrale ha affrontato lo sviluppo e l'implementazione di strategie avanzate nel campo dell'intelligenza artificiale, con particolare attenzione agli algoritmi di Reinforcement Learning (RL) applicati all'ottimizzazione delle traiettorie spaziali. L'obiettivo era di esplorare l'efficacia di algoritmi come Monte Carlo Tree Search (MCTS) e AlphaZero nella risoluzione di problemi di navigazione spaziale all'interno di ambienti complessi e dinamici.

La ricerca si è concentrata sull'integrazione di tecniche di Deep Learning con il framework MCTS per ottimizzare le traiettorie eliocentriche. L'intento era di superare i limiti delle tradizionali metodologie di pianificazione della missione, incrementando il Performance Index della missione in esame.

I risultati hanno evidenziato un potenziale miglioramento della soluzione trovata dal team del Politecnico di Torino nella *Global Trajectory Optimisation Competition*. In particolare l'approccio AlphaZero ha segnato un netto progresso rispetto alle metodologie più tradizionali, riuscendo a elevare il Performance Index della missione. Al contrario, l'approccio MCTS Vanilla, privo dell'integrazione con tecniche di Deep Learning, ha riscontrato risultati inferiori alle soluzioni trovate dal team del Politecnico.

L'analisi dei risultati evidenzia come l'approccio AlphaZero abbia ottenuto un vantaggio competitivo, dimostrando l'efficacia della fusione delle tecniche di Deep Learning con il framework MCTS. Questa sinergia ha permesso di affrontare con successo le sfide poste da ambienti altamente incerti e dinamici, tipici delle missioni spaziali interplanetarie.

I risultati ottenuti mostrano miglioramenti ed evidenziando l'innovatività dell'approccio utilizzato. L'integrazione di AlphaZero nel contesto dell'ottimizzazione delle traiettorie spaziali rappresenta un passo significativo avanti rispetto alle tradizionali tecniche di navigazione.

Nonostante i risultati promettenti, lo studio ha incontrato limitazioni legate alle sfide nell'implementazione degli algoritmi e nella potenza di calcolo a disposizione. Questi aspetti sottolineano l'importanza di continuare la ricerca per affinare ulteriormente le tecniche utilizzate.

I risultati hanno implicazioni significative per il campo dell'ingegneria aerospaziale, dimostrando il potenziale degli algoritmi di RL avanzati nel migliorare l'efficienza e l'efficacia delle missioni spaziali. Ricerche future potrebbero esplorare ulteriori algoritmi di *Reinforcement Learning* e l'applicazione di queste tecniche a nuovi problemi di ottimizzazione, possibilmente estendendo l'approccio a missioni reali.

Appendice A

Politecnico di Torino

A.1 Candidate_Asteroids.txt

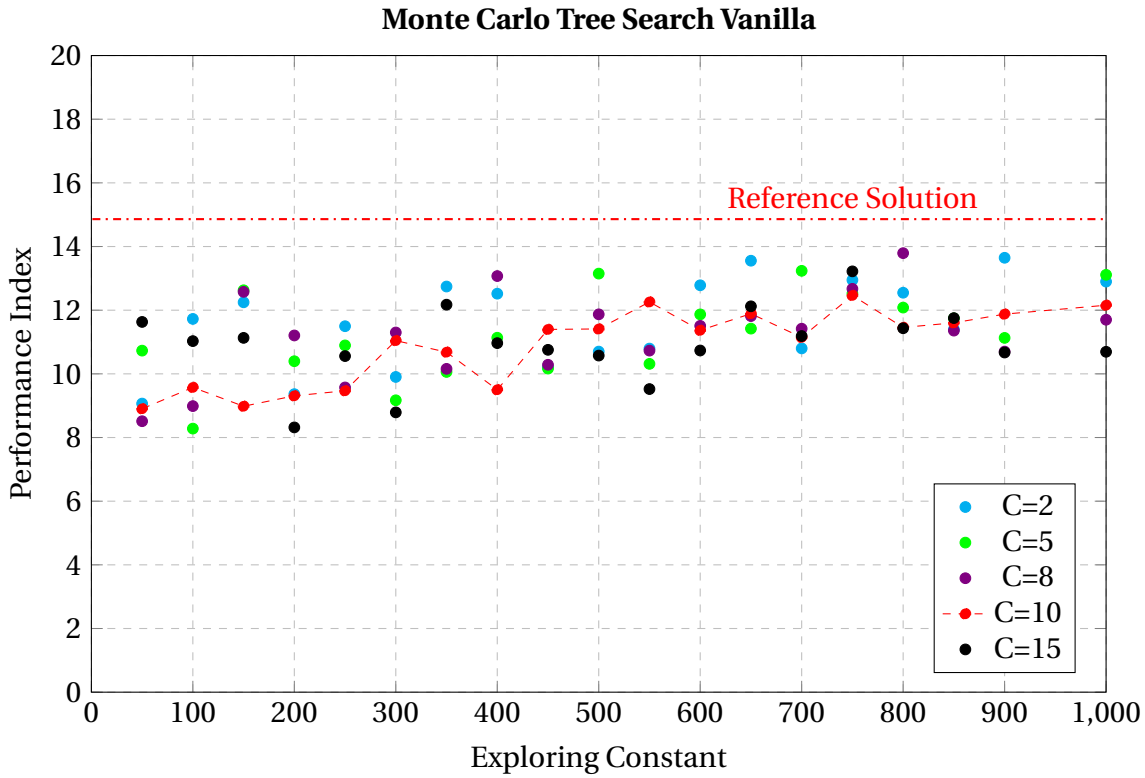
ID	MJD	a - AU	e	i - deg	Ω - deg	ω - deg	M - deg	Mass - kg
1	59396	3.2379003	0.06797738	13.97503429	178.138004	352.4606128	68.14966237	1.17043438082E+14
2	59396	3.17274146	0.17576680	9.593594651	217.005365	51.37307117	354.5474664	3.61715141814E+13
3	59396	2.72930491	0.37688847	8.45609843	33.7369904	28.3959106	98.63124884	1.38720711263E+13
4	59396	2.31396256	0.30484118	12.18476358	345.66619	3.517425755	213.9675733	4.64168409963E+12
5	59396	3.14683757	0.18582842	9.798373694	329.798089	149.957794	70.49796224	4.86305096734E+13
6	59396	3.12002989	0.46928616	24.25202019	46.8675695	102.1037516	29.664092	1.33355330266E+13
7	59396	2.91186697	0.11065434	3.81146797	153.0422637	276.547396	154.214458	2.22620920910E+13
8	59396	2.61227816	0.29995885	5.953880085	170.272330	202.3394456	215.7415817	3.14361519129E+13
9	59396	2.56820110	0.17242059	7.800011369	209.517325	112.4696286	260.2504476	7.03058558056E+12
10	59396	2.59683834	0.31312158	5.611258583	209.787031	188.8690552	195.8671883	2.66711172122E+12
...	...	...	...	...	...	...	...	...
48746	59396	2.62807816	0.32417226	30.57423136	320.86102	53.30367262	309.4238981	4.81673650087E+13
48747	59396	2.35587282	0.13599271	8.449271286	266.292318	247.433971	72.38245351	9.0832735344E+13
48748	59396	2.56054920	0.21871847	14.66938365	126.772579	238.5549471	4.641404651	2.97981240069E+13
48749	59396	2.23184134	0.21760312	6.249140734	290.383415	323.6206493	132.9869996	1.02388337782E+13
48750	59396	2.24857352	0.16920031	6.281833519	317.866029	314.2831244	103.5680517	7.18528735344E+12
48751	59396	2.56149472	0.23052701	14.99981646	123.966379	257.8654568	343.9479495	4.51841544485E+13
48752	59396	2.35599094	0.2244564	4.428815767	66.2764671	287.5425493	258.0889605	2.38724997111E+13
48753	59396	2.37054152	0.15913995	3.937077607	99.3198309	175.2539094	293.0438417	3.35667148860E+13
...	...	...	...	...	...	...	...	...
83445	59396	1.07811304	0.82700257	22.81198432	87.980551	31.41889942	180.7299314	1.57079632665E+12
83446	59396	2.33166489	0.24489450	9.225387409	292.802519	50.39235476	0.416879774	1.57731427989E+14
83447	59396	2.21558467	0.16602271	6.508279943	161.597475	224.049675	219.2975455	1.79424710260E+14
83448	59396	1.85353068	0.04457946	22.8241089	225.241945	340.2807342	71.23442185	1.22809783747E+14
83449	59396	1.91875859	0.43573296	11.88292067	171.3186	26.65518593	185.3656919	1.57079632666E+12
83450	59396	1.97886349	0.03945702	26.87117562	163.360956	349.676067	235.8791604	1.53394144133E+14
83451	59396	2.47366867	0.57045879	9.393753932	110.432768	350.507583	34.89945747	1.16377158256E+14
83452	59396	2.36106560	0.22682493	2.065913552	172.81587	190.0812282	327.0414339	2.54077592167E+13
83453	59396	2.24268055	0.17820379	4.234316005	95.0627113	123.4163269	168.7385926	1.42852728447E+14

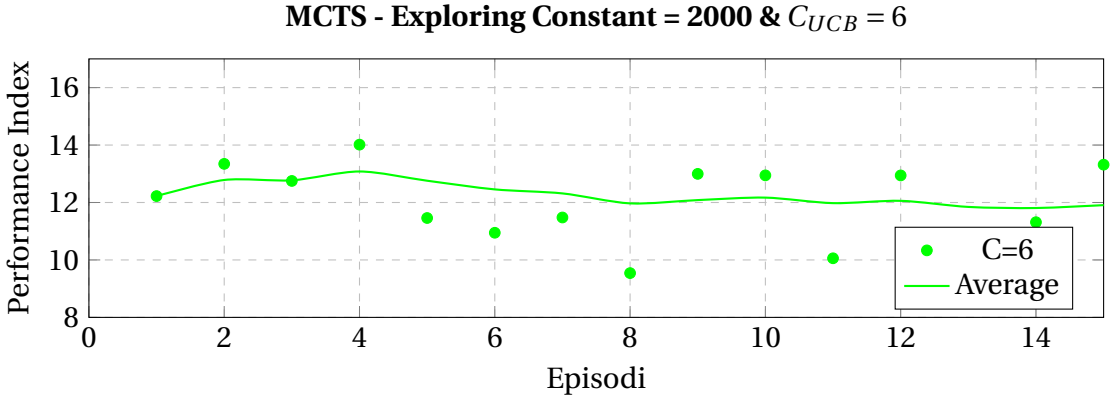
Table A.1: File.txt contenente i parametri orbitali e la masse dei 83453 asteroidi candidati per la risoluzione del problema. L'osservazione di questi parametri è relativa al 59396 MJD, ovvero il giorno 1 Luglio 2021.

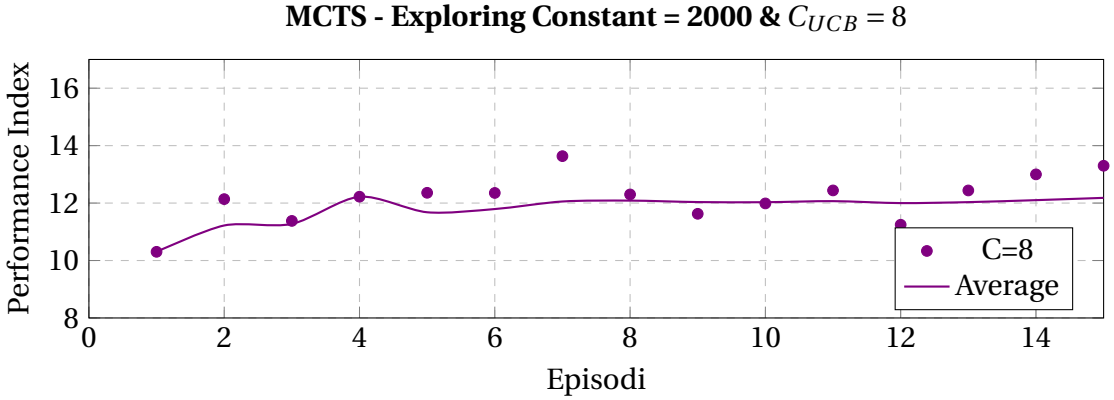
Politecnico di Torino

Appendice B

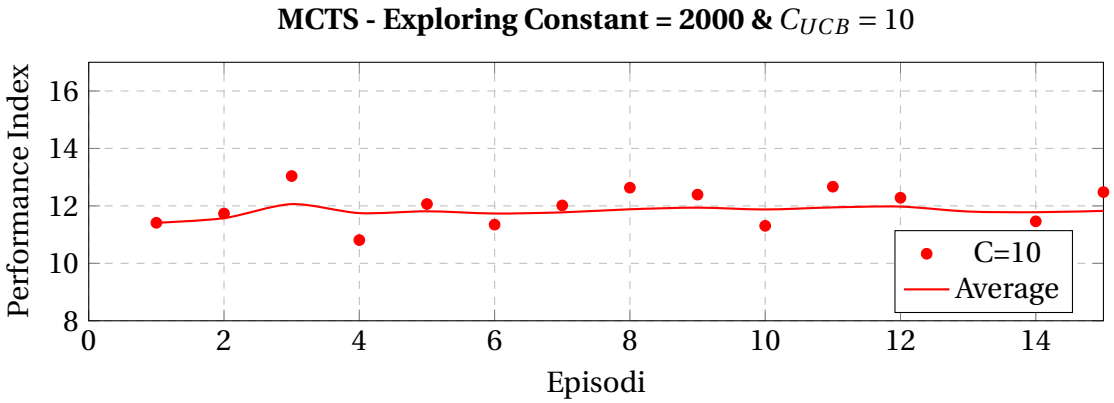
B.1 Simulazioni MCTS Vanilla

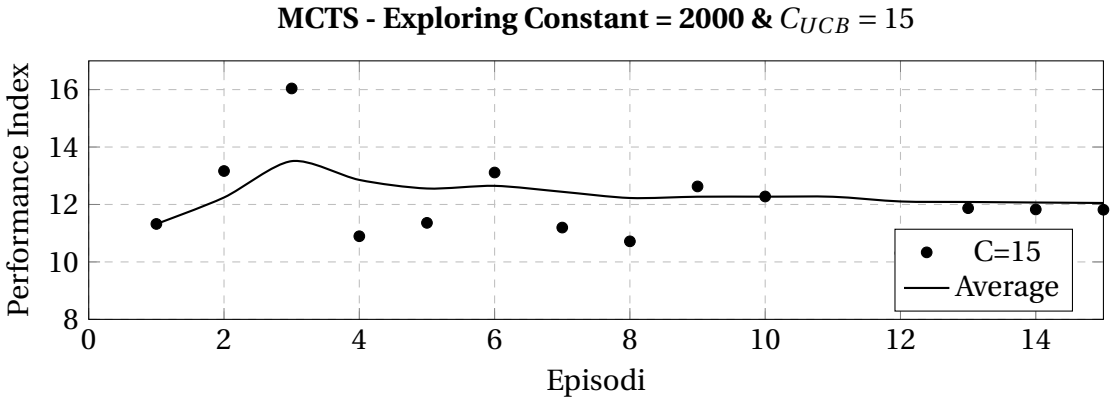


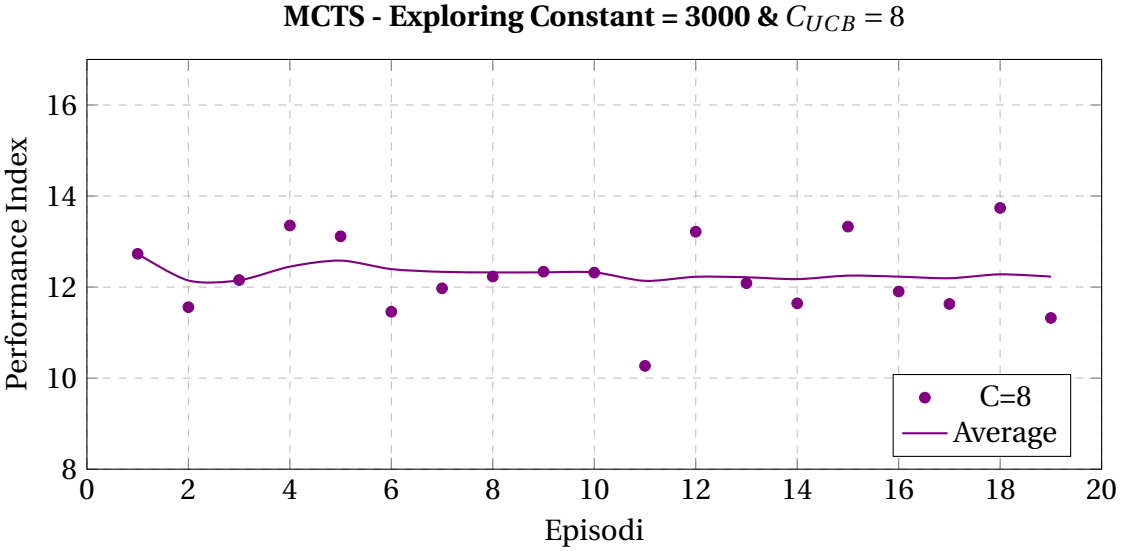




Politecnico di Torino







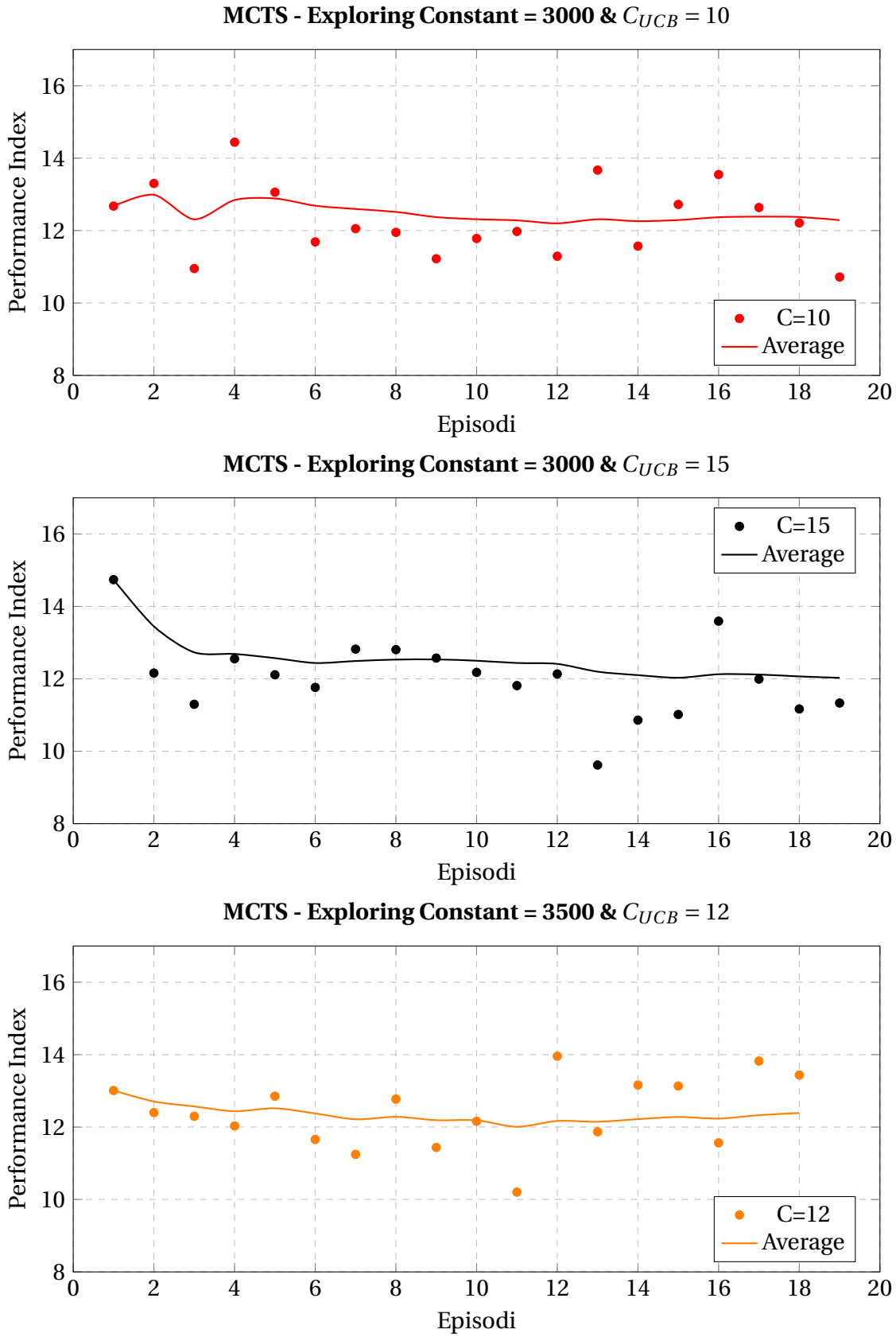


Figure B.1: MCTS Vanilla - Risultati delle simulazioni al variare del valore della costante UCB e dell'*exploring constant*.

BIBLIOGRAPHY

[1] Simon J.D Prince. *Understanding Deep Learning.* MIT Press, October 23, 2023.

[2] Richard S. Sutton and Andrew G.Barto. *Reinforcememt Learning - An Introduction.* The MIT Press, Cambridge, Massachusetts, 2020.

[3] Marco Kemmerling, Daniel Lütticke, and Robert H. Schmitt. Beyond games: A systematic review of neural monte carlo tree search applications, 2023.

[4] Ya-Zhong Luo, Yue-He Zhu, Peng Shu, Bing Yan, and Jia-Cheng Zhang. The problem of the 11^{th} grobal trajectory optimization competition - *Dyson Sphere Building*, 16 October 2021.

[5] Maciej Świechowski, Konrad Godlewski, Bartosz Sewicki, and Jacek Mańdziuk'. Monte carlo tree search: A review of recent modifications and applications, 2022.

[6] François Chollet. *Deep Learning with Python.* Manning Pubications Co., 2021.

[7] Eli Stevens, Luca Antiga, and Thomas Viehmann. *Deep Learning with PyTorch.* Manning Pubications Co., 2020.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org

[9] Ausélien Géeon. *Hands-On Machine Learning with Scikit-Learn, HìKeras & TensorFlow.* O'Reilly Media, 2019.

[10] Brian Ward. *How Linux works: What every superuser should know.* no starch press, 2021.

[11] William Shotts. *THE LINUX COMMAND LINE.* No Starch Press, 2012.

[12] Michaela K. Harlander. Introduction to unix. *Physik Department TU Munchen*, 1991.

[13] Roger R Bate, Donald D Mueller, Jerry E White, and William W Saylor. *Fundamentals of astrodynamics.* Courier Dover Publications, 1971.

[14] George P Sutton and Oscar Biblarz. *Rocket propulsion elements.* John Wiley & Sons, 2016.

[15] Bradley W Carroll and Dale A Ostlie. *An introduction to modern astrophysics.* Cambridge University Press, 2017.

[16] Howard D Curtis. *Orbital mechanics for engineering students.* Butterworth-Heinemann, 2013.

[17] Giovanni Mengali. Fondamenti di meccanica del volo spaziale. 2013.

[18] Hanspeter Schaub and John L Junkins. *Analytical mechanics of space systems.* Aiaa, 2003.

[19] Historic Buildings Demolition Row. Rocket propulsion and spaceflight dynamics. jw corne.

[20] Eric Matthes. *Python crash course: A hands-on, project-based introduction to programming.* no starch press, 2019.

[21] Daniel Zingaro. *Learn to Code by Solving Problems.* no starch press, 2021.

[22] Computational resources provided by hpc@polito. http://hpc.polito.it hpc@polito is a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino.

[23] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, and Timothy Lillicrap. Mastering atari, go, chess and shogi by planning with a learned model. *DeepMind, 6 Pancras Square, London*, 2020.

[24] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Crowling, Philipp Rohlfshanger, Stephen Tavener, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods, 2012.

[25] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Mattehew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhaeshan Kumran, Thore Graepel, Timothy Lillicrao, Karen Simonyab, and Demis Hassabis. Mastring chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[26] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van der Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[27] Qi Naiming, Fan Zichen, Huo Mingyning, Du Desong, and Zhao Ce. Fast trajectory generation and asteroid sequence selection in multispacecraft for multiasteroid exploration. *IEEE Trans Cybern*, 2022.

[28] Rita Neves. Monte carlo tree search for interplanetary trajectory design.

[29] Daniel Hennes and Dario Izzo. Interplanetary trajectory planning with monte carlo tree search. European Space Agency, 2015.

[30] Hattie Hall. Global optimisation of interplanetary trajectories. Master's thesis, University of Southampton, 2022.

[31] Dario Izzo. Revisiting lambert's problem. *Celestial Mechanics and Dynamical Astronomy*, 121:1–15, 2015.

[32] Ryan P Russell. On the solution to every lambert problem. *Celestial Mechanics and Dynamical Astronomy*, 131(11):50, 2019.

[33] Josh Varty. A step-by-step look at alpha zero and monte carlo tree search. https://joshvarty.github.io/AlphaZero/

[34] Int8. Monte carlo tree search - beginners guide. https://int8.io/monte-carlo-tree-search-beginners-guide/ 2018.

[35] Stanford Edu. A simple alpha(go) zero tutorial. https://web.stanford.edu/ surag/posts/alphazero.html 2017.

[36] Yazhe Niu, Yuan Pu, Zhenjie Yang, Xueyan Li, Tong Zhou, Jiyuan Ren, Shuai Hu, Hongsheng Li, and Yu Liu. Lightzero: A unified benchmark for monte carlo tree search in general sequential decision scenarios, 2023.

[37] Sotetsu Koyamada, Shinri Okano, Soichiro Nishimori, Yu Murata, Keigo Habara, Haruka Kita, and Shin Ishii. Pgx: Hardware-accelerated parallel game simulators for reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

[38] Sebastian Bodenstein. Alphazero. https://sebastianbodenstein.com/post/alphazero/ 2020.

[39] DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020.

Politecnico di Torino