REINFORCEMENT LEARNING

PARTE 1

INDICE

- REINFORCEMENT LEARNING APPLICATO AL TRADING
    1. MODULO TRADING SYSTEM
    2. MODULO ALLENAMENTO E VALUTAZIONE

- ANALISI DELL' APPLICAZIONE
    1. SCELTE DI IMPLEMENTAZIONE
    2. ANALISI DEI RISULTATI OTTENUTI

INTRODUZIONE

L'apprendimento per rinforzo o reinforcement learning è una tecnica di apprendimento automatico che punta a realizzare agenti autonomi in grado di scegliere azioni da compiere per il conseguimento di determinati obiettivi tramite l'interazione con l'ambiente in cui sono immersi.
Insieme all' apprendimento supervisionato e non supervisionato fa parte dell'apprendimento automatico ma si differenzia da questi perché tratta di problemi con decisioni sequenziali ovvero dove l'azione da compiere dipende dallo stato attuale del sistema e ne determina quello futuro.
Gli elementi principali di un problema di reinforcement learning sono:

- Una politica (policy), che definisce il modo di comportarsi dell'agente in un dato momento, ovvero una funzione che mappa gli stati dell'ambiente in azioni in modo tale che l'agente una volta che si trova un determinato stato, attraverso questa mappatura, sappia qual' è la migliore azione da intraprendere.

- Un segnale di ricompensa (reward signal), inviata ad ogni passo dall'ambiente all'agente sotto forma di valore numerico. Serve per far capire all'agente se le azioni che sta prendendo sono "buone" o "cattive" per il conseguimento del compito a lui assegnato.

- La ricompensa scontata (discounted reward), ottenuta pesando con opportuni coefficienti la reward signal attuale e quelle future. La massimizzazione di questa funzione definisce l'obiettivo principale dell'agente. Essa definisce la ricompensa totale che esso può aspettarsi qualora scelga un determinato stato e da questo gli stati che più probabilmente seguiranno.

PROCESSI DECISIONALI DI MARKOV

In questo capitolo viene introdotto in maniera formale il problema dell'apprendimento per rinforzo.

L'INTERFACCIA AGENTE-AMBIENTE

Il concetto base su cui si basa l'apprendimento per rinforzo è l'interazione agente-ambiente.
L'agente è colui che deve prendere la decisione mentre l'ambiente è tutto ciò che è al di fuori dall'agente, essi interagiscono continuamente, infatti l'agente seleziona ed esegue delle azioni e l'ambiente risponde presentando nuove situazioni o stati ed una ricompensa.
Più formalmente l'agente e l'ambiente interagiscono in ciascuna sequenza di passi temporali discreti t=0, 1, 2, 3… Ad ogni passo temporale t, l'agente riceve una rappresentazione dello stato dell'ambiente $S_t \in \mathbb{S}$, dove $\mathbb{S}$ è l'insieme dei possibili stati e su tale base seleziona un'azione $A_t \in \mathbb{A}(S_t)$, dove $\mathbb{A}(S_t)$ è l'insieme delle possibili azioni nello stato S_t.
Al passo successivo, come conseguenza della sua azione, l'agente riceve una ricompensa $R_{t+1} \in \mathbb{R} \subset \mathbb{R}$ finendo poi nello stato S_{t+1}.
Le azioni sono prese sulla base di una policy denominata $\pi_t(a|s)$ la quale indica la probabilità che $A_{t=}a$ dato $S_t=s$, semplificando al massimo si può dire che l'obiettivo dell'agente è quello di costruite una policy ottimale la quale gli permette di massimizzare la ricompensa attesa nel lungo periodo.
Il quadro fin ora esposto è astratto e flessibile nel senso che può essere applicato ad una varietà di problemi e in differenti modi, in generale le azioni possono essere tutte le decisioni che vogliamo che l'agente sia in grado di prendere e gli stati possono essere rappresentati da qualsiasi cosa che permetta all'agente di ricavare nel modo più efficace e efficiente possibile le informazioni necessarie per lo svolgimento del suo compito.
La regola generale che può essere usata per identificare il confine agente-ambiente è che tutto ciò che non può essere modificato arbitrariamente dall'agente fa parte dell'ambiente, in particolare questo confine è determinato una vota che si sono selezionati particolari stati, azioni e ricompense individuando cosi uno specifico compito decisionale d'interesse.

OBIETTIVI E RICOMPENSE

Nell'apprendimento per rinforzo, lo scopo o obiettivo dell'agente è formalizzato in termini di uno speciale segnale di ricompensa (un semplice numero $R_t \in \mathbb{R}$) che passa dall'ambiente all'agente, in particolare quello che dovrà fare è massimizzare la ricompensa cumulativa ovvero quella nel lungo periodo.
L'uso di un segnale di ricompensa per formalizzare l'idea di un obiettivo è una delle caratteristiche più distintive dell'apprendimento per rinforzo, di conseguenza è anche uno dei punti più cruciali che determinano il successo o meno del compito.
In particolare, se si vuole che l'agente faccia qualcosa per noi è necessario fornirgli ricompense in modo tale che, massimizzandole, possa raggiungere anche i nostri obiettivi
E' importante capire che il segnale di ricompensa non è uno strumento per impartire all'agente una conoscenza preliminare su come ottenere ciò che noi vogliamo, ad esempio, un agente che gioca a scacchi dovrebbe essere ricompensato solo per vincere, non per raggiungere obiettivi secondari come prendere determinate pedine del suo avversario o ottenere il controllo del centro della scacchiera, perché altrimenti potrebbe trovare un modo per raggiungere questi obiettivi di seconda importanza trascurando però quelli principali.

RENDIMENTO

Se la sequenza dei premi ricevuti dopo il passo temporale t si indica $R_{t+1,} R_{t+2,} R_{t+3} \ldots$ si cerca di massimizzare il rendimento atteso, dove il rendimento G_t è definito come una funzione che dipende dalla sequenza di ricompense.
Nel caso più semplice il rendimento è la somma delle ricompense:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T \qquad (1.0)$$

dove T è il passo temporale finale.
Questo approccio ha senso nelle applicazioni in cui esiste una nozione naturale di passo temporale finale, ovvero quando l'interazione dell'agente con l'ambiente si scompone naturalmente in sotto sequenze chiamate episodi, come per esempio negli scacchi.
Ogni episodio termina in uno stato speciale chiamato stato terminale e denotato con S^+, seguito da un ripristino ad uno stato iniziale standard o ad uno stato campionato da una distribuzione standard degli stati di partenza.
I compiti di questo tipo sono chiamati compiti episodici.
D'altra parte, in molti casi l'interazione agente-ambiente non si scompone naturalmente in episodi identificabili, ma va avanti continuamente senza limite.

Questi compiti sono chiamati continui.
La formulazione (1.0) è problematica perché il passo temporale finale T potrebbe tendere all'infinito e di conseguenza pure il ritorno.
Pertanto bisogna introdurre un concetto aggiuntivo che è quello di attualizzazione, ovvero, secondo questo approccio l'agente cerca di selezionare azioni in modo da massimizzare la somma delle ricompense future scontate.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + R_T = \sum \gamma^k R_{t+k+1}, \quad k=0\ldots\infty. \qquad (1.1)$$

Dove γ è un parametro chiamato tasso di sconto ed è compreso tra 0 e 1.

LA PROPRIETA' MARKOVIANA

Nell'apprendimento per rinforzo l'agente prende le sue decisioni in funzione di un particolare segnale fornito dall'ambiente che ne rappresenta lo stato.
Il segnale di stato può contenere sensazioni immediate (per esempio le rilevazioni dei sensori) o strutture più complesse costruite sulla base della sequenza di sensazioni acquisite nel tempo, per esempio, dato un panorama si può puntare lo sguardo ad ogni istante su un singolo punto ma si può anche costruire in dettaglio l'intero paesaggio sulla base delle osservazioni fatte.
Il concetto generale è quindi che la funzione di stato può contenere informazioni immediate, ricavate dal qui e ora oppure una ricostruzione delle stesse, non è importante che questa funzione informi l'agente di tutto ciò che riguarda l'ambiente, per esempio se l'agente gioca a black-jack non è necessario che sappia quale sia la prossima carta, se l'agente deve rispondere al telefono non è importante che sappia chi sia il chiamante.
Quello che si vorrebbe idealmente è una funzione di stato che riassuma la conoscenza passata in modo compatto e allo stesso tempo che conservi le informazioni rilevanti.
Una funzione che ha queste caratteristiche si dice che sia di Markov o che abbia la proprietà di Markov, per esempio la configurazione delle pedine di una scacchiera ha questa proprietà in quanto riesce a riassumere la storia passata della sequenza di stati che hanno portato a quella determinata configurazione, fornendo tutte le informazioni utili per il futuro.
Questa proprietà è anche chiamata "indipendenza dal percorso" perché tutto ciò che conta per il futuro sta nello stato attuale e non importa come ci si è arrivati.
Viene definita ora formalmente la proprietà di Markov per l'apprendimento per rinforzo, per mantenere la matematica semplice, si assume che ci sia un numero finito di stati e di valori di ricompensa.
Questo permette di lavorare in termini di somme e probabilità piuttosto che integrali e densità di probabilità, ma l'argomento può essere facilmente esteso nel caso continuo.

In generale la risposta dell'ambiente al passo t+1 in conseguenza dell'azione eseguita al passo t può dipendere da tutto ciò che è successo prima, in particolare questa dinamica è descritta dalla probabilità:

$$\Pr\{ R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1 \dots S_{t-1}, A_{t-1}, R_t, S_t, A_t \} \quad (1.2)$$

Per qualsiasi r e s' e per qualsiasi valore degli eventi passati S_0, A_0, R_1 … S_{t-1}, A_{t-1}, R_t, S_t, A_t.
Se invece il segnale di stato ha la proprietà di Markov la risposta dell'ambiente al passo t+1 è funzione solo dell'azione e dello stato precedente e la dinamica dell'ambiente può essere descritta da:

$$p(s', r \mid s, a) = \Pr\{ S_{t+1}=s' , R_{t+1} = r| S_t, A_t \} \quad (1.3)$$

Per qualsiasi r e s', S_t, A_t,

PROCESSI DECISIONALI DI MARKOV

Un compito di reinforcement learning che soddisfa la proprietà di markov è chiamato processo decisionale di Markov (o MDP), se lo spazio degli stati e delle azioni è finito viene chiamato processo decisionale di Markov finito.
Un particolare MDP finito è definito dagli insiemi dei suoi stati e azioni e dalla dinamica ad un passo dell'ambiente, ovvero dato un qualsiasi stato s e azione a la probabilità di ogni qualsiasi coppia stato-ricompensa (s', r) è data da:

$$p(s', r \mid s, a) = \Pr\{ S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\} \quad (1.4)$$

Data la (1.4) si può calcolare il premio previsto per la coppia stato-azione:

$$r(s, a) = E[R_{t+1} \mid S_t = s, A_t = a] = \sum_{r \in R} r \sum_{s' \in S} p(s', r \mid s, a) \quad (1.5)$$

La probabilità della transizione di stato:

$$p(s' \mid s, a) = \Pr\{ S_{t+1} = s' \mid S_t = s, A_t = a \} = \sum_{r \in R} p(s', r \mid s, a) \quad (1.6)$$

Le ricompense attese per la tripla stato-azione-stato successivo:

$$r(s, a, s') = E[R_{t+1} \mid S_t = s, A_t = a, S_{t+1}=s'] = \frac{\sum_{r \in R} r * p(s', r \mid s, a)}{p(s' \mid s, a)} \quad (1.7)$$

VALUE FUNCTION

Quasi tutti gli algoritmi di reinforcement learning implicano la stima della Value Function la quale fornisce un'indicazione di quanto è buono che l'agente si trovi in un determinato stato, dove per "quanto è buono" è inteso in termini di ricompense future. Informalmente il valore di uno stato s sotto la politica π indicato con v_π (s) è il rendimento atteso ottenuto a partire dallo stato s e seguendo la politica π.
Più formalmente:

$$v_\pi(s) = E_\pi[G_t \mid S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s] \quad (1.8)$$

Allo stesso modo si definisce il valore del prendere un'azione a in uno stato s sotto la politica π, indicato con $q_\pi(s, a)$ come il rendimento atteso a partire dallo stato s, compiendo l'azione a e seguendo poi la politica π:

$$q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a] \quad (1.9)$$

$q_\pi(s, a)$ è chiamata *action-value function*.
Una proprietà fondamentale della *value-function* è che soddisfi la seguente relazione ricorsiva:

$$v_\pi(s) = E_\pi[G_t \mid S_t = s] =$$

$$= E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s]$$

$$= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) [r + \gamma E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_{t+1} = s']]$$

$$= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) [r + \gamma v_\pi(s')] \quad (1.10)$$

Dove $a \in \mathbb{A}(S)$, lo stato attuale $s \in (\mathbb{S})$, lo stato successivo $s' \in (\mathbb{S} \cup \mathbb{S}^+)$ e $r \in \mathbb{R}$.
La (1.10) è l'equazione di Bellman per v_π ed esprime la relazione tra il valore di uno stato e il valore dei suoi stati successori, detto in altre parole quando l'agente si trova nello stato s la funzione v_π (s) ne rappresenta il rendimento atteso la quale viene calcolato sommando i premi di tutti i possibili scenari futuri ponderandoli per la rispettiva probabilità di avverarsi.

VALUE FUNCTION OTTIMA

Risolvere un problema di reinforcement learning, generalizzando, significa trovare una politica che permette di massimizzare le ricompense nel lungo periodo.
In particolare una politica π è migliore o uguale di una politica π' se il suo rendimento atteso è migliore o uguale di π' per tutti gli stati, ovvero $\pi \geq \pi'$ se e solo se $v_\pi(s) \geq v_{\pi'}(s)$ per ogni $s \in \mathbb{S}$.
La definizione sottintende che può esistere più di una politica ottimale, le quali verranno tutte rappresentate con la notazione π_*, esse condividono la medesima *value-function* la quale verrà chiamata in questo caso *value-function* ottima, indicata con v_* e definita come:

$$v_*(s) = max_\pi\, v_\pi(s) \qquad \text{per ogni } s \in \mathbb{S} \qquad (1.11)$$

Le politiche ottimali condividono anche la medesima *action-value function* ottima, indicata con q_* e definita come:

$$q_*(s, a) = max_\pi\, q_\pi(s, a) \qquad \text{per ogni } s \in \mathbb{S} \text{ e per ogni } a \in \mathbb{A}(S) \qquad (1.12)$$

Per la coppia stato-azione (s, a) questa funzione ritorna il rendimento atteso quando viene presa l'azione a nello stato s e successivamente viene seguita la politica ottimale, quindi si può scrivere q_* in termini di v_* come segue:

$$q_*(s, a) = E[R_{t+1} + \gamma\, v_*(S_{t+1}) \mid S_t = s, A_t = a] \qquad (1.13)$$

L'equazione (1.12) può essere scritta senza il riferimento esplicito alla politica π, assumendo così la forma:

$$\begin{aligned}
v_*(s) &= max_{a \in A(S)}\ q_{\pi^*}(s, a) \\
&= max_a\, E_{\pi^*}[\, G_t \mid S_t = s,\, A_t = a\,] \\
&= max_a\, E_{\pi^*}[\textstyle\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s,\, A_t = a\,] \\
&= max_a\, E_{\pi^*}[\, R_{t+1} + \gamma \textstyle\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s,\, A_t = a\,] \\
&= max_a\, E[\, R_{t+1} + \gamma\, v_*(S_{t+1}) \mid S_t = s,\, A_t = a\,] \\
&= max_{a \in A(S)}\ \textstyle\sum_{s',r} p(s', r \mid s, a)\, [\, r + \gamma\, v_*(s')\,]
\end{aligned} \qquad (1.14)$$

Mentre l'*action-value function* ottima assume la forma:

$$q_*(\text{s, a}) = \text{E}[\, R_{t+1} + \gamma\, max_{\text{a}'}\, q_*(\, S_{t+1}, \text{a}') \mid \; S_t = \text{s},\, A_t = \text{a}\,]$$
$$= \sum_{s',r}\, \text{p}(\text{s}', \text{r} \mid \text{s}, \text{a})\, [\, \text{r} + \gamma\, max_{\text{a}'}\, q_*(\text{s}', \text{a}')\,] \qquad (1.15)$$

Per gli MDP finiti, l'equazione (1.14) ha un'unica soluzione indipendente dalla politica. L'equazione di ottimalità di Bellman è in realtà un sistema di equazioni, una per ogni stato, quindi se ci sono N stati, allora ci saranno N equazioni in N incognite. Se la dinamica dell'ambiente è nota, allora in linea di principio si può risolvere questo sistema di equazioni, utilizzando uno dei qualsiasi metodi per risolvere sistemi di equazioni non lineari.
Una volta che si ha v_* è relativamente facile determinare una politica ottimale, infatti basta trovarne una, l'importante è che sia greedy rispetto a v_*.
Il termine greedy è utilizzato in informatica per descrivere qualsiasi procedura di ricerca o decisione che seleziona alternative basate solo su considerazioni locali o immediate, senza considerare la possibilità che tale selezione possa impedire l'accesso futuro ad alternative ancora migliori.
Il vantaggio di v_* è che il rendimento ottimale atteso a lungo termine viene trasformato in una quantità che è localmente e immediatamente disponibile per ogni stato.
Quindi, una ricerca ad un passo in avanti produce delle azioni che sono ottimali a lungo termine.

PROGRAMMAZIONE DINAMICA

Il termine programmazione dinamica (DP) si riferisce a una raccolta di algoritmi che possono essere utilizzati per calcolare le politiche ottimali dato un modello perfetto dell'ambiente, come un processo decisionale Markov.
Gli algoritmi DP classici sono di utilità limitata nell'apprendimento per rinforzo sia a causa della loro assunzione di un modello perfetto sia a causa della loro grande spesa computazionale, ma sono ancora importanti a livello teorico e in alcuni contesti pratici.
Per prima cosa viene mostrato come calcolare la funzione v_π per la politica arbitraria π, ciò viene chiamato *policy-evalutation*.
Se la dinamica dell'ambiente è completamente nota vuol dire risolvere un sistema di $|\mathbb{S}|$ equazioni in $|\mathbb{S}|$ incognite, in linea di principio il calcolo è semplice ma è computazionalmente oneroso, per questo vengono usati metodi iterativi.
Tipicamente il valore iniziale v_0 è scelto casualmente, dopo di che il valore della *value-function* viene aggiornato ad ogni passo attraverso l'equazione di Bellman nel seguente modo:

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) [\, r + \gamma\, v_k(s') \,]$$

Dove si può mostrare che la successione $\{v_k\}$ converge a v_π per $k \to \infty$.
L'algoritmo completo è il seguente:

Input: π, la politica
Inizializzare l'array V(s)=0 per ogni $s \in \mathbb{S}^+$.
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathbb{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) [\, r + \gamma\, V(s') \,]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
Until $\Delta < \theta$ (θ è un numero positivo piccolo a piacere)
Output $V \approx v_\pi$

MIGLIORAMENTO DELLA POLITICA

La ragione principale per cui viene calcolato il valore della *value-function* è quello di aiutare a trovare politiche migliori.
Supponendo di aver trovato il valore di v_π per un'arbitraria politica deterministica π, sarebbe interessante capire se per qualche stato s sarebbe più conveniente adottare un'azione diversa da quella proposta dalla politica, ovvero scegliere un'azione $a \neq \pi(s)$.
Una via per rispondere a questo quesito è quella di scegliere l'azione a e successivamente seguire la politica π, questo equivale a calcolare il valore:

$$q_\pi(s, a) = E_\pi[R_{t+1} + \gamma\, v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$$
$$= \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma\, v_\pi(s')] \qquad (1.16)$$

e poi verificare se è maggiore o minore di $v_\pi(s)$.
Se è migliore selezionare l'azione a nello stato s e poi seguire la politica π piuttosto che seguirla fin dall'inizio allora ci si può aspettare che è complessivamente migliore selezionare a ogni volta che ci si trovi nello stato s.
Questo è un caso speciale di un risultato più generale di un teorema chiamato *policy improvement.*
Siano π e π' due politiche deterministiche tali che per ogni $s \in \mathbf{S}$ vale:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \qquad (1.17)$$

allora la politica π' deve essere migliore o uguale a π, ovvero si deve ottenere un rendimento atteso migliore o uguale per tutti gli stati $s \in \mathbf{S}$, ovvero:

$$v_{\pi'}(s) \geq v_\pi(s) \qquad (1.18)$$

L'estensione naturale dei concetti fin qui esposti è quello di espandere la valutazione per ogni stato s e per ogni azione a, ovvero considerare la nuova politica greedy π' data da:

$$\pi'(s) = argmax_a\ q_\pi(s, a)$$
$$= argmax_a\ E[R_{t+1} + \gamma\, v_\pi(S_{t+1}) \mid S_t = s, A_t = a]$$
$$= argmax_a \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma\, v_\pi(s')] \qquad (1.19)$$

La politica π'(s) intraprende l'azione che sembra migliore a breve termine e per costruzione soddisfa le condizioni del teorema del miglioramento della politica (1.17) quindi è buona almeno quanto la politica originale.
Supponendo che la nuova politica π' è buona quanto ma non migliore della vecchia politica π, ovvero $v_{\pi'} = v_{\pi}$ dalla (1.19) segue che per ogni stato s∈**S**:

$$v_{\pi'}(s) = max_a \, E[\, R_{t+1} + \gamma \, v_{\pi'}(S_{t+1}) \mid S_t = s, \, A_t = a \,]$$
$$= max_a \sum_{s',r} p(s', r \mid s, a) \, [\, r + \gamma \, v_{\pi'}(s') \,] \qquad (1.20)$$

Ma questa è l'equazione di ottimalità di Bellman (1.14) e quindi $v_{\pi'} = v_*$, ovvero entrambe le politiche sono ottimali.
Nel caso generale di una politica stocastica definita come π(a|s) il teorema del miglioramento della politica viene applicato alla quantità:

$q_{\pi}(s, \pi'(s)) = \sum_a \pi'(a \mid s) \; q_{\pi}(s, a)$.

POLICY ITERATION

Una tecnica che può essere usata per il miglioramento della politica è chiamata *policy iteration* la quale permette di ottenere una successione di politiche, via via sempre più efficaci fino al raggiungimento di una politica ottimale.
Una volta che una politica π è stata migliorata attraverso l'uso di v_π per produrre una politica migliore π', quest'ultima può essere a sua volta usata insieme a $v_{\pi'}$ per produrre una politica ancora migliore π'' e cosi via dando luogo alla successione:

$$\pi_0 \xrightarrow{E} v_{\pi 0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi 1} \xrightarrow{I} \dots \pi_* \xrightarrow{E} v_*$$

dove E sta per *evalutation* e I sta per *improvement*.
L'algoritmo è il seguente:

1. Inizialization
   V(s)∈ $\mathbb{R}$ e π(s)∈ $\mathbb{A}$(S) arbitrariamente per ogni $s \in \mathbb{S}$

2. Policy Evalutation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathbb{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) [r + \gamma V(s')]$
   $\Delta \leftarrow \max (\Delta , |v - V(s)|)$
   Until $\Delta < \theta$ (θ è un numero positivo piccolo a piacere)

3. Policy Improvement
   Politica-stabile ← true
   For each $s \in \mathbb{S}$:
   $a \leftarrow \pi(s)$
   $\pi(s) \leftarrow argmax_a \sum_{s',r} p(s', r \mid s, a) [r + \gamma V(s')]$
   if $a \neq \pi(s)$ then *Politica-stabile* ← false
   if *Politica-stabile*, then stop e ritorna V e π else go to 2.

L'unico bug di questo algoritmo è che potrebbe non fermarsi mai se si alternano due politiche ugualmente valide, ma questo può essere facilmente risolto aggiungendo un paio di flag.
Inoltre essendo che un MDP finito ha un numero finito di policy questo algoritmo converge ad una politica ottimale in un numero finito di iterazioni.

VALUE ITERATION

Uno svantaggio della *policy iteration* è che ciascuna iterazione implica la valutazione della politica, che può essere essa stessa un calcolo iterativo oneroso, oltre al fatto che la convergenza avviene soltanto al limite.
L'algoritmo *value iteration*, combinando le fasi di valutazione e miglioramento in un'unica fase permette di troncare in maniera evidente il numero di iterazioni garantendo comunque la convergenza.
La regola di aggiornamento è la seguente:

$v_{k+1}(s) = max_a \sum_{s',r} p(s', r \mid s, a) [r + \gamma v_k(s')]$

Mentre l'algoritmo completo è:

Inizializzare l'array V(s) arbitrariamente (es. V(s)=0 per ogni $s \in \mathbb{S}^+$)
Repeat
$\Delta \leftarrow 0$
For each $s \in \mathbb{S}$:
$v \leftarrow V(s)$
$V(s) \leftarrow max_a \sum_{s',r} p(s', r \mid s, a) [r + \gamma V(s')]$
$\Delta \leftarrow \max (\Delta , |v - V(s)|)$
Until $\Delta < \theta$ (θ è un numero positivo piccolo a piacere)
Output, una politica deterministica tale che:
$\pi(s) = argmax_a \sum_{s',r} p(s', r \mid s, a) [r + \gamma V(s')]$

METODI MONTE CARLO

A differenza della programmazione dinamica qui non si assume una conoscenza preliminare completa dell'ambiente, i metodi Monte Carlo richiedono solo l'esperienza ovvero sequenze di campioni di stati, azioni e ricompense ottenute da un'iterazione reale o simulata.
Questi metodi risolvono il problema dell'apprendimento per rinforzo basandosi sulla media dei rendimenti campionari e per garantire che questi rendimenti siano ben definiti vengono trattate solo attività episodiche.
L'efficacia di questi metodi è che permettono di raggiungere comunque un comportamento ottimale sebbene la (1.2) sia ignota.

PREDIZIONE MONTE CARLO

In questo paragrafo viene mostrato come stimare la *value-function* per una data politica. L'idea base di tutti i metodi MC è semplicemente quella di fare la media di tutti i rendimenti osservati dopo le visite ad un determinato stato, man mano che si osservano più rendimenti la media dovrebbe convergere al valore atteso.
Si definisce visita a s ogni occorrenza dello stato s che è raggiunta all'interno di un episodio, mentre prima visita se è la prima volta che si visita s.
In particolare supponendo di voler stimare v_π(s) dato un insieme di episodi ottenuti seguendo π e passando per s si parla di metodo *first-visit* se la stima viene fatta come media dei rendimenti dopo le prime visite a s e di metodo *every-visit* se la stima viene calcolata come media dei rendimenti dopo tutte le visite a s.
Entrambi i metodi convergono purchè il numero di visite o prime visite a s sia grande, infatti essendo che ogni rendimento è una stima indipendente, identicamente distribuita e con varianza finita per la legge dei grandi numeri la sequenza delle medie di queste stime converge al valore atteso.
In questo capitolo verrà trattato solo il primo metodo il cui algoritmo è descritto dal seguente codice:

Inizializzazione
$\pi \leftarrow$ la politica da valutare
V $\leftarrow$ un'arbitraria *value-function*
Return(s) $\leftarrow$ una lista vuota per ogni $s \in \mathbf{S}$

Repeat
Generare un episodio usando π
Per ogni stato s che appare nell'episodio:
    G ← rendimento seguendo la prima visita a s
    Appendere G a Return(s)
    V(s) ← media(Return(s))

Una caratteristica di questo algoritmo ma anche dei metodi MC in generale è che le stime per ogni stato sono indipendenti e non si basano sulla stima di un altro stato come avviene nei metodi DP, questo vuol dire che la spesa computazionale per la stima di un singolo stato potrebbe essere di molto inferiore rispetto ai metodi mostrati nei capitoli scorsi, il che rende i metodi Monte Carlo molto attrattivi quando bisogna stimare un solo valore o sottoinsieme di valori di stati.

STIMA MONTE CARLO DELLA *ACTION-VALUE*

Se un modello non è disponibile è particolarmente utile stimare i valori delle azioni piuttosto che quelli degli stati, infatti con un modello i valori degli stati sono sufficienti a determinare una politica, basta semplicemente guardare avanti di un passo e scegliere l'azione che porta alla migliore combinazione di ricompensa e stato successivo esattamente come viene fatto nei metodi DP.
Al contrario senza un modello si deve stimare esplicitamente il valore di ogni azione affinchè sia utile nel suggerire una politica, in altre parole bisogna stimare la quantità q_π (s, a).
I metodi MC per far ciò sono sostanzialmente uguali a quelli per stimare la *value-function* tranne che ora si parla di visite alla coppia stato-azione piuttosto che ad uno stato, dove si dice che una coppia stato-azione (s,a) sia visitata in un episodio se lo stato s viene visitato e in esso viene intrapresa l'azione a.
L'unico problema è che molte coppie stato-azione potrebbero non essere mai visitate in quanto seguendo π ogni volta che si incontra uno stato verrà sempre scelta una sola delle sue azioni, quella migliore secondo la politica e di conseguenza le stime Monte Carlo delle altre azioni non miglioreranno con l'esperienza.
Un modo per far si che ogni coppia stato-azione venga valutata è quello di specificare che tutti gli episodi iniziano con una coppia stato-azione e che ogni coppia ha una probabilità diversa da zero di essere selezionata come inizio, questo garantisce che tutte le coppie verranno visitate un numero infinito di volte al limite di un numero infinito di episodi. Questo è chiamato presupposto dell'inizio dell'esplorazione.

Questo approccio è utile ma non sempre praticabile come per esempio quando si apprende dall'interazione diretta con l'ambiente, infatti è poco probabile che le condizioni di partenza siano cosi utili.
L'approccio alternativo consiste nel considerare solo politiche stocastiche con una probabilità diversa da zero di selezionare tutte le azioni in ogni stato.

MONTE CARLO CONTROL

In questo capitolo viene spiegato come la stima Monte Carlo possa essere impiegata nel controllo, ovvero per approssimare le politiche ottimali.
L'idea base è quello di procedere secondo lo stesso schema della *policy-iteration* ovvero si mantiene sia una politica approssimata sia una *value-function* approssimata, quest'ultima viene valutata secondo la politica corrente mentre la politica viene migliorata rispetto alla nuova *value-function,* creando così un loop che permetterà di avvicinarsi all'ottimalità.
Per il momento si assumono le ipotesi che vengono realmente osservati un numero infinito di episodi e che questi siano generati con il presupposto dell'inizio dell'esplorazione.
Nel caso dei metodi MC la fase di miglioramento avviene rendendo la politica avida rispetto la *action-value* e quindi non è necessario predisporre di nessun modello. Per ogni funzione q la politica greedy corrispondente è quella che per ogni $s \in \mathbf{S}$ sceglie deterministicamente un'azione con valore massimo:

$\pi(s) = argmax_a \; q(s, a)$

Il miglioramento della politica può essere fatto così costruendo ogni π_{k+1} come politica greedy rispetto q_{π_k}, inoltre dati π_k e π_{k+1} si ha:

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, argmax_a \; q_{\pi_k}(s, a)) \\ &= max_a \; q_{\pi_k}(s, a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &= v_{\pi_k} \end{aligned}$$

Ovvero vale il teorema del miglioramento della politica il quale assicura che ogni π_{k+1} è almeno buona quanto π_k il che significa che il processo complessivo converge verso la politica ottimale.

Per raggiungere questo risultato sono state fatte due ipotesi, la prima è che la valutazione della politica poteva essere fatta con un numero infinito di episodi, ipotesi al quanto irrealistica nella realtà.
Ci sono due modi per risolvere questo problema, il primo è quello di non approssimare q_{π_k} ad ogni valutazione, effettuando ipotesi per ottenere limiti sull'entità e sulla probabilità di errore delle stime e garantire per ogni valutazione che questi limiti siano sufficientemente piccoli.
Il secondo consiste nel rinunciare a completare la fase di valutazione prima di tornare al miglioramento delle politiche, una forma estrema di questo approccio è quando viene eseguita una sola iterazione della valutazione della politica ad ogni fase del processo.
Per la valutazione delle politiche Monte Carlo è naturale alternare tra valutazione e miglioramento episodio per episodio. Dopo ogni episodio, i rendimenti osservati vengono utilizzati per la fase di valutazione della politica, quindi la politica viene migliorata in tutti gli stati visitati nell'episodio. Un semplice algoritmo con "*exploring starts"* è il seguente:

Inizializzare per ogni $s \in \mathbb{S}$ e $a \in \mathbb{A}(S)$:
Q(s, a) ← arbitrariamente
π(s) ← arbitrariamente
Return(s, a) ← una lista vuota

Repeat forewer:
scegliere $S_0 \in \mathbb{S}$ e $A_0 \in \mathbb{A}(S_0)$ tali che tutte le coppie hanno probabilità > 0
generare un episodio che inizia con S_0 e A_0, seguendo π
per ogni coppia (s, a) che appare nell'episodio:
G ← rendimento seguendo la prima occorrenza di (s, a)
Appendere G a Return(s, a)
Q(s, a) ← media(Return(s, a))
Per ogni s nell'episodio:
$\pi(s) \leftarrow argmax_a$ Q (s, a)

La seconda ipotesi su cui si basava il risultato precedente è che valesse il presupposto dell'inizio dell'esplorazione, per rimuovere tale assunzione si possono considerare due metodi: i *on-policy* e i *off-policy*.
I primi usano sempre la stessa politica mentre i secondi migliorano una politica diversa rispetto a quella che è stata usata per generare i dati. Un esempio di approccio *on-policy* è l'algoritmo Monte Carlo appena illustrato.

Nei metodi di controllo *on-policy* la politica è generalmente *soft*, questo significa che $\pi(a \mid s) > 0$ per tutti gli $s \in \mathbb{S}$ e $a \in \mathbb{A}(S)$ ma gradualmente si sposta sempre più vicino ad una politica ottimale. Quelli considerati in questa sezione si basano invece su una politica *ε-greedy* (sottoinsieme delle politiche ε-soft ma più avide) la quale sceglie la maggior parte delle volte un'azione che ha il valore massimo ma con probabilità ε sceglie un'azione a caso, cioè a tutte le azioni non avide viene data la probabilità $\frac{\varepsilon}{|A(S)|}$ e la rimanente probabilità $1 - \varepsilon + \frac{\varepsilon}{|A(S)|}$ è data all'azione greedy. Le politiche *ε-greedy* sono quindi esempi di politiche *ε-soft* definite per $\pi(a \mid s) \geq \frac{\varepsilon}{|A(S)|}$ per tutti gli stati e azioni e per un $\varepsilon > 0$.
Per rimuovere il presupposto dell'inizio dell'esplorazione basterà quindi che la politica sia portata verso una politica avida e non esattamente ad una politica avida, in altre parole significa migliorare la politica fino a che raggiunga un livello *ε-greedy*.
Per una qualsiasi politica *ε-soft* π, e qualsiasi politica *ε-greedy* π' che rispettano q_π è garantito che quest'ultima è migliore o uguale a π, infatti per ogni $s \in \mathbb{S}$ vale che:

$$
\begin{aligned}
q_\pi(s, \pi'(s)) &= \sum_a \pi'(a \mid s)\ q_\pi(s, a) \\
&= \frac{\varepsilon}{|A(S)|} \sum_a q_\pi(s, a) + (1 - \varepsilon)\, max_a\, q_\pi(s, a) \\
&\geq \frac{\varepsilon}{|A(S)|} \sum_a q_\pi(s, a) + (1 - \varepsilon) \sum_a \frac{\pi(a \mid s) - \frac{\varepsilon}{|A(S)|}}{1 - \varepsilon} q_\pi(s, a) \\
&= \frac{\varepsilon}{|A(S)|} \sum_a q_\pi(s, a) - \frac{\varepsilon}{|A(S)|} \sum_a q_\pi(s, a) + \sum_a \pi(a \mid s) q_\pi(s, a) \\
&= v_\pi(s)
\end{aligned}
$$

Ovvero che per il teorema del miglioramento della politica si ha che $\pi' \geq \pi$ per tutti gli stati s.
Un algoritmo è:

Inizializzare per ogni $s \in \mathbb{S}$ e $a \in \mathbb{A}(S)$:
Q(s, a) ← arbitrariamente
Return(s, a) ← una lista vuota
π(a | s) ← un'arbitraria politica *ε-soft*

Repeat forewer:
(a) generare un episodio usando π
(b) per ogni coppia (s, a) che appare nell'episodio:
G ← rendimento seguendo la prima occorrenza di (s, a)
Appendere G a Return(s, a)
Q(s, a) ← media(Return(s, a))

(c) Per ogni s nell'episodio:

a* ← $argmax_a$ Q(s, a)

per ogni a∈ 𝔸(S):

$$\pi(a \mid s) \leftarrow \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A(S)|} & \text{se a} = a* \\ \frac{\varepsilon}{|A(S)|} & \text{se a} \neq a* \end{cases}$$

Il secondo approccio che consente di eliminare il presupposto dell'inizio dell'esplorazione come accennato in precedenza sono i metodi *off-policy*.
In questo contesto quello che si ha a disposizione sono degli episodi generati da una politica diversa rispetto a quella che si vuole ottimizzare, ovvero si vuole stimare v_π e quello che si ha a disposizione sono episodi generati da una politica μ , con $\mu \neq \pi$, dove π è la politica obiettivo mentre μ è chiamata politica comportamento in quanto controlla l'agente e ne definisce il comportamento.
Per poter usare episodi generati da μ per stimare valori di π è necessario richiedere che ogni azione intrapresa sotto π sia almeno occasionalmente intrapresa sotto μ, questo equivale a dire in altre parole che $\pi(a \mid s) > 0$ implichi $\mu(a \mid s) > 0$, questa è chiamata ipotesi di copertura. Da questa ipotesi ne segue che μ deve essere stocastico negli stati in cui non è identica a π, quest'ultima invece può essere deterministica e questo è un caso di particolare interesse.
Tipicamente π è una politica avida deterministica rispetto all'attuale stima della *action-value* function, quello che ci si augura è che questa diventi una politica ottimale deterministica, mentre μ rimane stocastica e più esplorativa per esempio una *ε-greedy*.
In questo contesto la stima di $v_\pi(s)$ viene calcolata attraverso la tecnica di Importance Sampling la quale permette di stimare il valore atteso di una distribuzione attraverso i campioni di un'altra, in particolare si ha:

$$V(s) = \frac{\sum_{t \in \Gamma(s)} \rho_t^{T(t)} G_t}{|\Gamma(s)|} \tag{1.21}$$

Dove $\Gamma(s)$ rappresenta l'insieme di tutti i passi temporali in cui viene visitato s e $\rho_t^{T(t)}$ sono i rapporti di importance sampling cosi definiti:

$$\rho_t^T = \frac{\prod_{k=t}^{T-1} \pi(A_k \mid S_k) p(S_{k+1} \mid S_k, A_k)}{\prod_{k=t}^{T-1} \mu(A_k \mid S_k) p(S_{k+1} \mid S_k, A_k)} = \frac{\prod_{k=t}^{T-1} \pi(A_k \mid S_k)}{\prod_{k=t}^{T-1} \mu(A_k \mid S_k)}$$

Mentre $\prod_{k=t}^{T-1} \pi(A_k \,|S_k) p(S_{k+1} \,|S_k, A_k)$ rappresenta, dato lo stato iniziale S_t, la probabilità della successione $A_t, S_{t+1}, A_{t+1}, \ldots, S_T$ sotto la politica π e di conseguenza ρ_t^T rappresenta la probabilità relativa della traiettoria (successione) in relazione alla politica comportamento e obiettivo.
In questa tecnica di importance sampling i rendimenti sono scalati per ρ_t^T , in realtà i metodi *off-policy* utilizzano il campionamento di importanza ponderato ovvero

$$V_n = \frac{\sum_{k=1}^{n-1} W_k G_k}{\sum_{k=1}^{n-1} W_k} \qquad n \geq 2 \qquad (1.22)$$

Dove $G_1, G_2 \ldots G_{n-1}$ è una sequenza di rendimenti ottenuta partendo sempre dello stesso stato e dove a ciascun rendimento k è associato un peso W_k (es. $W_k = \rho_t^T$).
Affinchè questo algoritmo incrementale funzioni bisogna tenere traccia, oltre che dei V_n, la somma cumulativa dei pesi per i primi n rendimenti di ogni stato denotata C_n.
La regola di aggiornamento è la seguente:

$$V_{n+1} = V_n + \frac{W_n}{C_n} [\, G_n - V_n \,] \qquad \text{e} \qquad C_{n+1} = C_n + W_{n+1} \qquad n \geq 1 \qquad (1.23)$$

dove $C_0 = 0$ e V_1 è arbitrario.
Un algoritmo che si basa sui metodi off-policy è il seguente:

Inizializzare per ogni $s \in \mathbb{S}$ e $a \in \mathbb{A}(S)$:
Q(s, a) ← arbitrariamente
C(s, a) ← 0
π(s) ← una politica deterministica avida rispetto a Q

Repeat forewer:
generare un episodio usando una qualsiasi politica soft μ
$S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T S_T$
G ← 0
W ← 1
For t = T-1, T-2,… downto 0:
G ← γG + R_{t+1}
$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [\, G - Q(S_t, A_t)]$
$\pi(S_t) \leftarrow argmax_a \; Q(S_t, a)$
$W \leftarrow W \frac{1}{\mu(A_t, S_t)}$
If W = 0 then exit for loop

TEMPORAL DIFFERENCE LEARNING

L'apprendimento per differenza temporale è una combinazione di idee Monte Carlo e di Programmazione Dinamica, infatti l'agente può imparare direttamente dall'esperienza senza bisogno di un modello delle dinamiche dell'ambiente (come in MC) e può aggiornare le sue stime senza attendere un risultato finale (come in DP).

PREDIZIONE TEMPORAL DIFFERENCE

Sia il metodo Monte Carlo che Temporal Difference usano l'esperienza per risolvere il problema della previsione.
I metodi MC aspettano fino a quando non si conosce il ritorno successivo alla visita e quindi usano quel ritorno come obiettivo per $V(S_t)$:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Ovvero devono attendere fino alla fine dell'episodio per determinare l'incremento di $V(S_t)$.
I metodi TD invece devono attendere solo fino al passaggio temporale successivo, infatti all'istante t+1 formano immediatamente un target ed effettuano un'utile aggiornamento usando la ricompensa osservata R_{t+1} e la stima $V(S_{t+1})$:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Infatti l'obiettivo per l'aggiornamento MC è G_t mentre per TD è $R_{t+1} + \gamma V(S_{t+1})$.
In altre parole data:

$$v_\pi(s) = E_\pi[\, G_t \mid S_t = s] = \tag{1.24}$$

$$= E_\pi\left[\, \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \,\right]$$

$$= E_\pi\left[\, R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s \,\right]$$

$$= E_\pi[\, R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \,] \tag{1.25}$$

i metodi Monte Carlo usano una stima di (1.20) mentre i metodi di Programmazione Dinamica usano una stima di (1.21) come obiettivo.
Si parla di stima perché in MC il valore atteso (1.20) non è noto e viene usato un rendimento campione piuttosto che il rendimento atteso reale, invece di DP l'obiettivo è una stima non a causa dei valori attesi che vengono forniti direttamente dalla dinamica dell'ambiente ma perché $v_\pi(S_{t+1})$ non è noto e viene usata la sua stima corrente $V(S_{t+1})$.
L'obiettivo in TD è una stima per entrambi i motivi: campiona i valori attesi in (1.21) e usa la stima corrente V al posto di v_π, motivo per il quale questi metodi sono una combinazione di idee di quelli visti nei capitoli precedenti.
Un algoritmo che contiene i concetti spiegati è il seguente:

Input: π, la politica da valutare
Inizializzare l'array V(s) arbitrariamente (es. V(s)=0 per ogni $s \in \mathbb{S}^+$)
Repeat (per ogni episodio):
    Inizializzare S
    Repeat (per ogni passo dell'episodio):
        A ← azione data da π per S
        Prendi l'azione A, osserva la ricompensa R, e il prossimo stato S'
        V(S) ← V(S) + α[R + γV(S') - V(S)]
        S ← S'
    Finche S è terminale

VANTAGGI DEI METODI DI PREVISIONE TD

I metodi a differenza temporale apprendono le loro stime sulla base di altre stime, ovvero fanno il bootstrap. In questa sezione verrà analizzato il fatto se sia buono o meno questo tipo di metodologia e in particolare i vantaggi rispetto agli altri due metodi.
Un primo vantaggio rispetto ai metodi di programmazione dinamica è che non richiedono la conoscenza di un modello dell'ambiente.

Il secondo vantaggio più ovvio rispetto invece ai metodi Monte Carlo è che sono implementati in modo on-line ovvero non bisogna aspettare la fine di un episodio per la fase di aggiornamento dei valori.
La domanda più importante che ci si può porre adesso è se questi metodi assicurano la convergenza, la risposta fortunatamente è si infatti è stato dimostrato che per ogni politica π l'algoritmo sopra descritto converge a v_π nella media di un parametro *step-size* costante se questo è sufficientemente piccolo.
Arrivati a questo punto la prossima domanda che ci si può porre è se i metodi TD convergono prima rispetto ai metodi MC, sfortunatamente a questa domanda non si è ancora riusciti a rispondere in modo formale, l'unica cosa che si può dire è che in pratica sono più veloci dei metodi MC nella risoluzione di compiti stocastici.

TEMPORAL DIFFERENCE CONTROL

Come in Monte Carlo i metodi per risolvere il problema del controllo si dividono in due classi: i *on-policy* e gli *off-policy*.
Il primo passo per entrambi è quello di imparare la *action-value* function, in particolare per un metodo *on-policy* bisogna stimare q_π (s, a) , per l'attuale politica π e per tutte le coppie stato-azione, di fatto per questo task può essere usato lo stesso metodo TD che è stato implementato per l'apprendimento di v_π, infatti se in precedenza venivano considerate le transizioni da stato a stato ora la differenza è che le transizioni vanno da una coppia stato-azione ad un'altra coppia stato-azione. Formalmente i due casi sono identici perché si tratta sempre di catene di Markow con un processo di ricompensa.
Un semplice algoritmo che fa parte di questa classe di metodi è chiamato SARSA ed è il seguente:

Inizializzare Q(s, a) per ogni $s \in \mathbb{S}$, $a \in \mathbb{A}(S)$ arbitrariamente e Q(stato terminale,a)=0
Repeat (per ogni episodio):
    Inizializzare S
    Scegli A da S usando la politica derivata da Q (es. *ε-greedy*)
    Repeat (per ogni passo dell'episodio):
        Prendi l'azione A, osserva la ricompensa R, e il prossimo stato S'
        Scegli A' da S' usando la politica derivata da Q (es. *ε-greedy*)
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma\, Q(S',A') - Q(S,A)]$
        $S \leftarrow S'$; $A \leftarrow A'$
    Finche S è terminale

Come tutti i metodi *on-policy* questo algoritmo cerca di stimare continuamente q_π per la politica di comportamento π e allo stesso tempo viene modificato π in modo tale da renderla avida rispetto a q_π .
Una delle scoperte più importanti nell'apprendimento per rinforzo è stato lo sviluppo di un algoritmo TD fuori politica chiamato Q-LEARNING, la regola di aggiornamento che lo contraddistingue è data da:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \, max_a \, Q(S_{t+1}, a) - Q(S_t, A_t)]$$

In questo caso Q approssima direttamente q_* (la politica ottimale) indipendentemente dalla politica seguita, questo semplifica di molto l'analisi dell'algoritmo il quale ha dimostrato di convergere con probabilità 1 a q_* .
Il codice completo è il seguente:

Inizializzare Q(s, a) per ogni $s \in \mathbb{S}$, $a \in \mathbb{A}(S)$ arbitrariamente e Q(stato terminale,a)=0
Repeat (per ogni episodio):
    Inizializzare S
    Scegli A da S usando la politica derivata da Q (es. *ε-greedy*)
    Repeat (per ogni passo dell'episodio):
        Prendi l'azione A, osserva la ricompensa R, e il prossimo stato S'
        Scegli A' da S' usando la politica derivata da Q (es. *ε-greedy*)
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \, max_a \, Q(S', a) - Q(S,A)]$
        $S \leftarrow S'$;
    Finche S è terminale

APPLICAZIONI DEL REINFORCEMENT LEARNING

Le applicazioni dell'apprendimento per rinforzo in passato erano limitate da un'infrastruttura informatica debole. Tuttavia, i progressi nell' ambito di nuove tecnologie computazionali stanno aprendo la strada ad applicazioni stimolanti completamente nuove.
Alcuni degli ambiti in cui il Reinforcement Learning ha avuto maggiore successo includono:

- Gioco
- Robotica
- Consigli personalizzati
- Gestione delle risorse

Il gioco è probabilmente il campo di utilizzo più comune per l'apprendimento per rinforzo e dove è stato in grado anche di ottenere prestazioni maggiori, riuscendo perfino a raggiungere livelli "sovraumani".
Quelli dove si ha raggiunto il maggiore successo sono: i giochi Atari, Go (attraverso AlphaGo) e poker (con DeepStack/Libratus), ognuno di questi rappresenta una grande famiglia di problemi e le tecniche sottostanti possono essere applicate ad un gran numero di applicazioni.
Per esempio AlphaGo lo è per giochi a somma zero con informazioni perfette per i due giocatori, mentre DeepStack/Libratus lo è per i giochi a somma zero con informazioni imperfette, entrambi hanno ottenuto risultati straordinari su un problema molto difficile da risolvere e perciò sono diventati un punto di riferimento nell'AI.
Per capire meglio perché vengono definiti straordinari i risultati raggiunti in questo ambito può essere preso in considerazione il gioco Go, dove due giocatori, usando pietre bianche o nere, a turno le posizionano su un tabellone. L'obiettivo è circondare e catturare le pietre dell'avversario o creare strategicamente spazi di territorio. A partita conclusa vengono conteggiate sia le pietre sul tabellone che i punti vuoti. Vince il numero più alto.
Per quanto semplici possano sembrare le regole, Go è profondamente complesso, infatti ci sono 10^{170} possibili configurazioni di schede cioè più del numero di atomi nell'universo conosciuto. Questo rende il gioco molte volte più complesso degli scacchi.
AlphaGo, l'intelligenza artificiale che gioca a GO, combina un albero di ricerca avanzato con reti neurali profonde. Queste reti neurali prendono una descrizione della

scheda Go come input e la elaborano attraverso una serie di diversi livelli di rete contenenti milioni di connessioni simili a neuroni.
Una rete neurale, la "rete politica", seleziona la mossa successiva da giocare. L'altra rete neurale, la "rete del valore", prevede il vincitore del gioco.
Nel corso del tempo, AlphaGo è migliorato ed è diventato sempre più forte e migliore nell'apprendimento e nel processo decisionale, continuando a sconfiggere i campioni del mondo di Go in diverse arene globali e probabilmente diventandone il più grande giocatore di tutti i tempi.
La robotica è un altro ambito di applicazione tradizionale per RL e può essere applicato ad un'ampia gamma di aree, ad esempio produzione, catena di approvvigionamento, assistenza sanitaria, ecc.
Questo settore pone sfide importanti al reinforcement learning, tra cui dimensionalità (numero delle dimensioni trattate nel modello), sottomodellazione (modelli che non catturano tutti i dettagli della dinamica del sistema), incertezza del modello, progettazione della ricompensa e specifica dell'obiettivo.
RL fornisce i seguenti approcci trattabili alla robotica:

- Rappresentazione: che include la discretizzazione dello stato delle azioni, l'approssimazione della funzione del valore e le politiche prestrutturate.
- Conoscenze pregresse: che includono la dimostrazione e la strutturazione dei compiti e della direzione dell'esplorazione.
- Modelli: che tengono conto di bias di simulazione, stocasticità del mondo reale ed efficienza di ottimizzazione con campioni di simulazione.

Gli algoritmi RL sono generalmente ad alta intensità di campionamento e l'esplorazione potrebbe generare politiche rischiose per il robot e/o l'ambiente, è quindi preferibile addestrare un robot in un ambiente di simulazione piuttosto che nella realtà. Tuttavia, di solito c'è un divario tra simulazione e realtà rispetto alla dinamica e alla percezione (acquisizione dai sensori) del robot, quindi un simulatore non può riflettere con precisione la realtà e tutti i dettagli che la caratterizzano. Si può comunque tentare di ridurre questo divario per ottenere un migliore risultato nei seguenti modi:

- migliorando il simulatore analiticamente o utilizzando dati, ma di solito questo processo non è banale e racchiude molte difficoltà.
- progettare una politica solida per le proprietà del sistema, assumendo una simulazione imperfetta.

Un modo per ottenere la robustezza è la randomizzazione, ad esempio, utilizzando una politica stocastica, randomizzando le dinamiche, aggiungendo rumore alle osservazioni o perturbando il sistema con disturbi casuali.
Per alcuni problemi di RL, le funzioni di ricompensa potrebbero non essere disponibili o difficili da specificare, si può pensare quindi di usare l'apprendimento per imitazione,

dove un agente impara a svolgere un compito da dimostrazioni di esperti senza segnali di rinforzo. La clonazione comportamentale e la RL inversa sono i due approcci principali di questa tecnica.
Di seguito un esempio di applicazione di RL per un dexterous robot (robot manipolatore).
I robot sono comunemente utilizzati per operazioni strutturate e automatiche come quelle che vengono fatte nelle fabbriche. Tuttavia, è auspicabile progettare manipolatori per eseguire compiti, ad esempio spostare oggetti, in ambienti non strutturati come le case e gli ospedali, in situazioni simili il robot deve sapersi muovere in un ambiente dinamico e non conosce a priori la posizione degli oggetti, a differenza di come potrebbe capitare in un ambiente statico quale una fabbrica.
In questo contesto un oggetto, ad esempio un blocco o un prisma ottagonale, viene posizionato sul palmo di un robot umanoide e l'obiettivo è quello di riorientare l'oggetto su una configurazione target, a tale scopo viene usata una mano robotica umanoide Shadow Dexterous Hand.
Le osservazioni sono: le posizioni della punta delle dita, la posizione dell'oggetto, l'orientamento dell'oggetto, l'orientamento del target, l'orientamento relativo del target, gli angoli delle articolazioni della mano, la velocità delle articolazioni della mano, la velocità dell'oggetto e la velocità angolare dell'oggetto. Le azioni sono gli spostamenti degli angoli dei giunti.
Ad ogni passo temporale, c'è una differenza di angoli di rotazione tra l'orientamento desiderato dei giunti dell'oggetto e quello corrente. La ricompensa è definita per ogni transizione, come la differenza del valore assunto dagli angoli prima e dopo la transizione, inoltre, viene assegnata una ricompensa di 5 per il raggiungimento di un obiettivo e una penalità di -20 per la caduta dell'oggetto.

Ci sono quattro passaggi per apprendere un'abile manipolazione manuale:

1. fare esperienza
2. formare una politica utilizzando RL
3. prevedere la posa dell'oggetto
4. combinare la stima della posa e la politica da implementare sul robot

Nella fase 1, vengono raccolti i dati per apprendere la politica e per stimare le pose basate sul sistema di visione. Molti aspetti dell'ambiente simulato sono randomizzati, inclusi la fisica, i rumori e parte delle osservazioni e del sistema visivo. Nel passaggio 2, viene imparata una politica basata sull'esperienza utilizzando RL, per esempio con una rete neurale ricorrente (RNN).
Nel passaggio 3, si impara a prevedere la posa dell'oggetto dalle immagini, per esempio con una rete neurale convoluzionale (CNN).

Nel passaggio 4 viene implementata la politica e la stima della posa appresa con la simulazione nel robot reale, in cui le posizioni dei polpastrelli vengono misurate utilizzando un sistema di acquisizione del movimento 3D e vengono utilizzati tre feed di telecamere per prevedere la posa dell'oggetto.
La politica può apprendere vari tipi di presa, in particolare presa di presa di punta, presa di presa palmare, presa di treppiede, presa di precisione a 5 dita e presa di potenza.
Vengono ora prese in considerazione le applicazioni del reinforcment learning ai sistemi di raccomandazione.
Un sistema di raccomandazione può suggerire prodotti, servizi, informazioni, ecc. agli utenti in base alle loro preferenze, li aiuta a prendere decisioni attraverso consigli personalizzati.
Nei sistemi di raccomandazione incentrati sull'utente, è essenziale comprendere e soddisfare le loro reali esigenze e preferenze, utilizzando un'interazione naturale, discreta e trasparente con essi. Questi sistemi devono stimare, suscitare, reagire e influenzare lo stato latente dell'utente (es. soddisfazione, preferenze, bisogni, interessi, processi comportamentali che governano il processo decisionale e il ragionamento dell'utente ecc.) mediante l'interazione naturale con essi e agendo nel migliore interesse per loro.

In questo contesto RL incontra alcune sfide:

- gestire il numero enorme di utenti e le azioni che possono potenzialmente compiere.
- natura idiosincratica delle azioni, come insiemi di azioni stocastiche e dimensioni dinamiche dello spazio di stato.
- stato latente dell'utente, che porta a un alto grado di non osservabilità e stocasticità.

I banditi contestuali sono efficaci per selezionare azioni basate su informazioni contestuali, ad esempio nel proporre articoli di notizie, selezionando articoli in base a informazioni contestuali come le attività storiche degli utenti e le informazioni descrittive del contenuto dell’articolo. Questo framework copre un gran numero di applicazioni, tuttavia tale impostazione incontra due sfide:

1. Feedback parziale. La ricompensa è solo per l'azione selezionata, ma non per le azioni inesplorate.
2. Premi ritardati. Le informazioni sulla ricompensa possono arrivare molto più tardi rispetto a quando si è intrapreso un'azione.

Di conseguenza, un'implementazione di solito deve affrontare le seguenti problematiche:

- feedback e bias parziali
- errata raccolta dei dati
- cambiamenti negli ambienti
- monitoraggio e debug deboli

Il test A/B è un approccio di esplorazione per affrontare il problema del feedback parziale.
L'A/B testing è usato principalmente nel marketing online, serve ad ottimizzare il tasso di conversione delle proprie campagne marketing, cioè a trasformare un numero maggiore di visitatori in acquirenti o abbonati**.**
E' un metodo attraverso il quale è possibile testare due diverse versioni dello stesso sito web (la versione A e la B), o alcuni suoi elementi (ad es. una landing page, ma anche titoli o layout), inviandoli a due gruppi di user differenti, in altre parole, è una sorta di "esperimento" che i webmaster fanno per determinare quale versione funziona meglio.
La tecnica dei Banditi contestuali consente di testare e ottimizzare un numero esponenziale di politiche con la stessa quantità di dati e non richiede l'implementazione di policy per il test e questo permette di risparmiare lavoro e attività di progettazione, tale capacità è denominata test multimondo (MWT).
L'implementazione della tecnica dei banditi contestuali viene di fatto poi implementata attraverso l'architettura "Decision Service" la quale fornendo la capacità di MWT aiuta a risolvere le 4 problematiche prima esposte. Essa definisce quattro astrazioni di sistema, vale a dire, esplorare per raccogliere dati, registrare i dati correttamente, apprendere un buon modello e distribuire il modello nell'applicazione usando tecniche di banditi contestuali e valutazione delle politiche.
Il servizio contribuisce inoltre a rendere il sistema reattivo, riproducibile, scalabile, tollerante ai guasti e flessibile per l'implementazione. Il servizio Decision ha diverse opzioni di distribuzione: self-hosted nell'account Azure di un utente e modello locale sul computer di un utente.
Infine viene trattato il problema di gestione delle risorse prendendo in considerazione come esempio di riferimento il raffreddamento di un data center discutendo quindi il problema del consumare energia in modo efficiente.
Il raffreddamento è essenziale per l'infrastruttura del data center per abbassarne la temperatura elevata e per ridurne la quantità di calore al fine di migliorare le prestazioni e mitigare i potenziali danni alle apparecchiature che si possono verificare qualora non siano raffreddate adeguatamente.

Esistono difficoltà come eventi imprevisti, vincoli di sicurezza, dati limitati e guasti potenzialmente costosi.
Si discute quindi di come controllare la velocità della ventola e il flusso d'acqua nelle unità di trattamento dell'aria (AHU) per regolare il flusso d'aria e la temperatura all'interno dei server, utilizzando il controllo predittivo del modello (MPC).
In MPC, il controller (agente) apprende un modello lineare delle dinamiche del data center con un'esplorazione casuale e sicura, con poca o nessuna conoscenza preliminare, quindi ottimizza il costo (ricompensa) di una traiettoria in base al modello previsto e genera azioni a ogni passaggio per mitigare l'effetto dell'errore del modello e dei disturbi imprevisti.
I controlli o le azioni sono variabili da manipolare, inclusa la velocità della ventola per controllare il flusso d'aria e l'apertura della valvola per regolare la quantità d'acqua. Gli stati fanno riferimento alle variabili di processo per prevedere e regolare, tra cui la pressione dell'aria differenziale (DP), la temperatura del corridoio freddo (CAT), la temperatura dell'aria in entrata (EAT) in ciascuna AHU, la temperatura dell'aria in uscita (LAT) da ciascuna AHU. Ci sono anche disturbi del sistema, riferiti a eventi o condizioni non manipolabili o controllabili, inclusi l'utilizzo dell'energia del server per surrogare la quantità di calore generato e la temperatura dell'acqua in ingresso (EWT) che viene refrigerata in ciascuna AHU.
Il metodo viene confrontato con un controllore derivato integrale proporzionale locale (PID) che aggiorna i parametri del modello dinamico. Gli esperimenti dimostrano che il metodo può ottenere il raffreddamento del data center in un sistema commerciale su larga scala in modo sicuro, efficace ed economico.

PUNTI DEBOLI DEL REINFORCEMEN LEARNING

Come si può aver capito dagli esempi sopra esposti, il reinforcement learning può essere utile in una molteplicità di situazioni, ma sicuramente non è la soluzione a tutto.
Il modo in cui l'apprendimento per rinforzo modella il problema del mondo reale richiede diverse condizioni che non sempre sono specificabili o di facile definizione.
Un esempio può essere quello della definizione della ricompensa che guida l'agente nella direzione "giusta".
Questa funzione deve catturare esattamente ciò che si vuole che l'agente raggiunga come obiettivo finale.
Non sono rari i casi in cui l'agente, nonostante abbia massimizzato la ricompensa attesa, non abbia raggiunto gli obiettivi precedentemente imposti, questo perché nella sua fase di esplorazione è riuscito a trovare una strada alternativa che gli permettesse di ottenere un maggior guadagno nel lungo termine ma che non prevedeva però il raggiungimento degli obiettivi definiti dal ricercatore.
Ecco perché i giochi Atari sono un punto di riferimento ideale per RL, perché non solo forniscono un'ampia dimensione del campione di dati, ma anche perché la definizione della funzione di ricompensa è molto semplice da progettare infatti l'obiettivo di ogni partita è solo quello di massimizzare il punteggio.
Sfortunatamente non tutti i problemi del mondo reale possono essere risolti in modo così semplice, basti pensare ad un agente che deve eseguire la pianificazione del percorso per un veicolo autonomo, in questo caso la funzione di ricompensa non è banale ne dal punto di vista matematico ne dal punto di vista della progettazione, infatti non è ovvio come tradurre ad un agente il fatto che deve raggiungere una determinata destinazione.
Un approccio per aggirare questo problema è l'apprendimento per rinforzo inverso il quale si concentra maggiormente sul progettare una buona funzione di ricompensa piuttosto che sul modello o sulla politica come invece viene fatto in quello tradizionale.
Un altro punto fondamentale è che gli agenti imparano sempre dalla combinazione di esplorazione e sfruttamento, infatti RL è un apprendimento continuo basato su tentativi ed errori, in cui l'agente cerca di applicare diverse combinazioni di azioni su uno stato per trovare la più alta ricompensa cumulativa.
L'esplorazione diventa quasi impossibile in molti problemi del mondo reale.
Basta considerare per esempio il caso in cui si vuole far imparare al robot a navigare in ambienti complessi evitando collisioni, man mano che il robot si muove nell'ambiente per apprendere, esplorerà nuovi stati e intraprenderà diverse azioni per navigare, tuttavia non è possibile intraprendere sempre le azioni migliori nel mondo reale in cui la dinamica dell'ambiente cambia molto frequentemente e questo implica che il robot fallirà diverse volte prima di trovare la politica corretta ma questi fallimenti potrebbero essere troppo costosi da sostenere.

Quindi per evitare il problema di cui sopra, sono stati applicati diversi altri meccanismi agli agenti RL per farli apprendere, nel caso della robotica una tecnica abbastanza diffusa è l'apprendimento per imitazione, dove esperti insegnano manualmente i movimenti al robot e nel caso della guida autonoma possono essere usati ambienti di simulazione.
Un'assunzione che viene fatta nei modelli di reinforcement learning è che il processo che va a descrivere il modello sia Markoviano, ma questo non è vero in generale, basta pensare al mondo in cui viviamo per capire che questa è una semplificazione un pò troppo riduttiva.
Inoltre il mondo è solo parzialmente osservabile quindi il più delle volte l'agente potrebbe avere accesso ad informazioni imprecise e incomplete e di conseguenza necessiterà di maggiore esplorazione implicando un maggior consumo di risorse in termini di tempo, risorse computazionali ecc.
Da ultimo bisogna considerare che il mondo è riccamente strutturato in oggetti complessi e le loro relazioni reciproche sono importanti, quindi non può essere modellato semplicemente come un vettore di caratteristiche statiche.
Se bisognasse progettare un sistema di robot che operino insieme per il raggiungimento di un obiettivo comune il problema potrebbe diventare molto più complesso e il reinforcement learning non sempre garantirebbe una soluzione sicura.
Detto questo sicuramente il reinforcement learning è una delle novità più importanti degli ultimi anni ma come tale ha bisogno di essere ulteriormente approfondita o integrata con altre tecniche come infatti è successo dando luogo ad alcune varianti interessanti come per esempio il reinforcemet learnig gerarchico o il deep reinforcemet learnig.

PARTE 2

REINFORCEMENT LEARNING APPLICATO AL TRADING

Questo capitolo è dedicato alla spiegazione del codice relativo all' implementazione di un sistema di trading basato su reinforcement learning.
Il programma si compone di 3 moduli:

1. TradingSystem
2. Allenamento
3. Valutazione

Il modulo Allenamento contiene la classe TradingEnv la quale fornisce i dati e le funzioni utili per l'implementazione dell'algoritmo Q-LEARNING implementato nel modulo Allenamento, infine il modulo Valutazione usa la q-table prodotta in fase di allenamento per testare l'agente.

MODULO TRADING SYSTEM

Il sistema è composto da 3 stati:

1. OUT: se non c'è nessun trade in corso, ovvero si è fuori dal mercato.
2. SHORT: se c'è una posizione corta aperta.
3. LONG: se c'è una posizione lunga aperta.

Le azioni che può compiere l'agente sono 2:

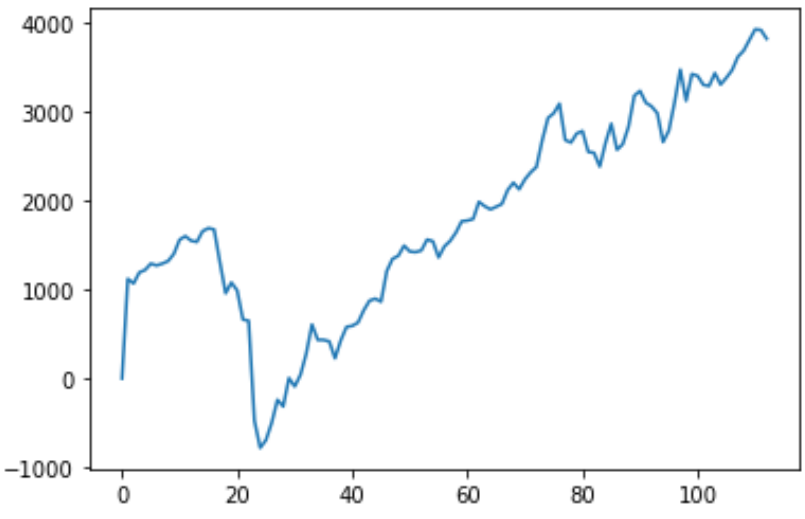
1. SELL: ovvero una vendita allo scoperto.
2. BUY: un aquisto.

Una prima assunzione che è stata fatta è che al massimo si può acquistare/vendere un solo contratto alla volta e quindi non è possibile incrementare la propria posizione, questo vuol dire che se l'agente esegue l'azione SELL quando il suo stato è SHORT rimane in quello stato con il contratto che aveva venduto antecedentemente, in modo analogo se l'agente esegue l'azione BUY mentre è nello stato LONG l'effetto di tale azione è di far restare l'agente nel suddetto stato con il contratto che è stato precedentemente acquistato.

L’effetto di un’operazione BUY o SELL in uno stato OUT è quello di far entrare l’agente a mercato, al contrario un’azione BUY in uno stato SHORT implica l’uscita dal mercato come l’effetto di un’azione SELL in uno stato LONG ha lo stesso effetto.
Quello detto fin’ora è descritto dalla funzione get_state() della classe TradingEnv.
Per questo sistema si è pensato di usare come ricompensa la differenza del prezzo di entrata e quello di uscita in quanto rappresenta l’effettivo guadagno o perdita subita, ciò viene calcolato dalla funzione calculate_reward().
Le altre funzioni di questo modulo sono la reset() che viene usata all’inizio di ogni episodio per riinizializzare gli stati, la action_sample() la quale ha il compito di campionare le azioni in maniera casuale usando la distribuzione uniforme e la step() che definisce il passaggio da uno stato all’altro dell’algoritmo.

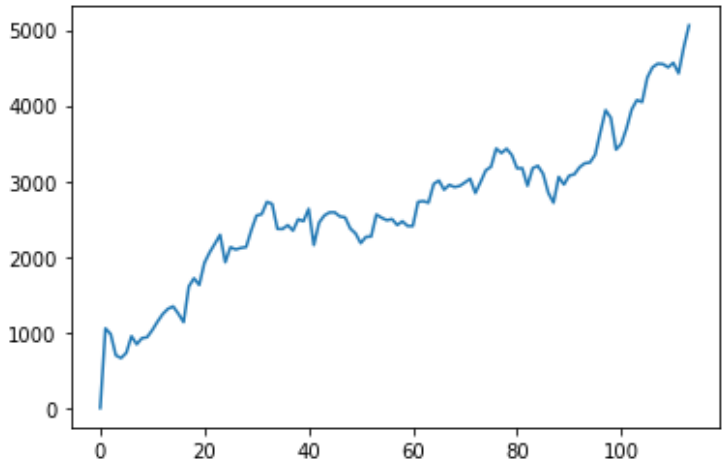
MODULI ALLENAMENTO E VALUTAZIONE

Il modulo allenamento contiene l’implementazione del codice Q-LEARNING descritto nella parte 1 di conseguenza non c’è bisogno di una descrizione dettagliata del codice, l’unica nota è che è stato scelto un valore per γ di 0.6 perché è il valore che permetteva di avere i risultati migliori.
Allo stesso modo è stato usato per il parametro window_size il valore di 200, che corrisponde appunto a 200 candele ovvero circa i dati di due giornate quando si utilizza un Time Frame di 15 minuti, questo perché permettesse di raccogliere movimenti relativamente lunghi e che si estendono per più di un giorno.
Il modulo valutazione contiene il codice usato per testare l’agente.
La prima prova che è stata fatta consiste nell’allenare l’agente su dati per un periodo complessivo di 2 anni, da febbraio 2017 a dicembre 2019 ed eseguendo una prima fase di valutazione sui dati dell’anno 2020 producendo il seguente andamento del capitale:



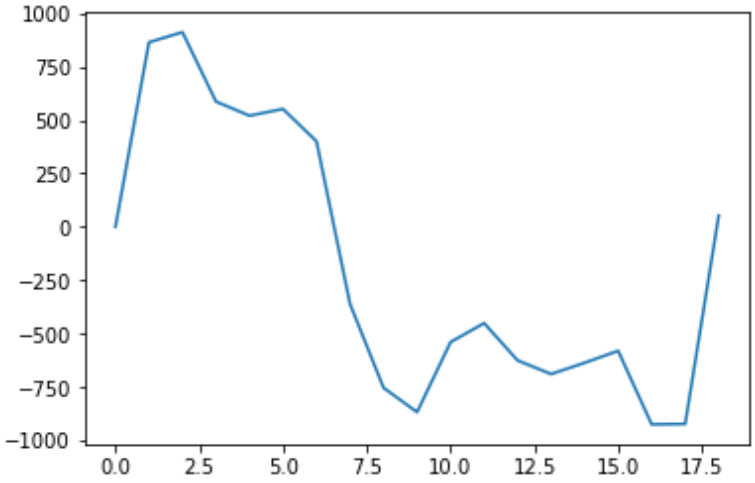
(fig.1)

E sull'anno 2021 producendo il seguente risultato:



(fig.2)

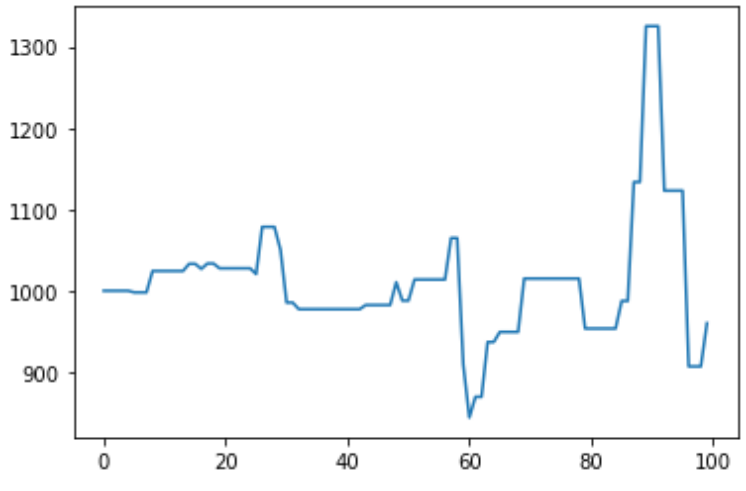
Dove nel calcolo di questi dati si è assunto un capitale iniziale di 1000 euro.
Un'interpretazione di ciò sta nel fatto che il NASDAQ 100 negli ultimi anni è stato soggetto ad un trend positivo, infatti sia i dati di allenamento che quelli di valutazione potevano essere interpolati su di una trend line crescente.
Per constatare ciò è stata fatta una prova anche per l'anno 2022 dove a causa dell'incremento dei tassi d'interesse proposto dalla FED e dalla guerra il mercato ha subito una forte fase ribassista per poi muoversi lateralmente.
Sull'anno 2022 è stato prodotto il seguente risultato:



(fig.3)

Infatti come ci si aspettava l'agente non è stato in grado di attuare una strategia vincente a causa del cambiamento del movimento di fondo del mercato.
Si è poi deciso di riproporzionare in maniera diversa la fase di allenamento con quella di valutazione, in particolare, sempre su un Time Frame a 15 minuti è stato per prima

cosa cambiato il valore del parametro window_size portandolo a 96, ovvero i dati contenuti in una giornata e poi si è deciso di alternare un periodo di allenamento composto da 15 episodi (giorni) ad un giorno di valutazione.
Questo è stato implementato attraverso i moduli TradingSystem2 e versione2.
Il risultato di ciò è rappresentato nel seguente grafico:



(fig.4)

Un'interpretazione di ciò può essere dovuta al fatto che l'agente non è stato allenato per un numero sufficientemente alto di episodi e quindi non è stato in grado di imparare una strategia ottimale per il breve periodo.

ANALISI DELL'APPLICAZIONE

SCELTE DI IMPLEMENTAZIONE

Per prima si discute l'ambiente dove viene addestrato l'agente, ovvero i mercati azionari.
Questo è un ambiente stocastico ovvero ogni dato può essere interpretato come un elemento di una successione di variabili casuali, che però non possono essere descritte tramite dei processi noti come per esempio MA, AR, ARMA, ARMAX ecc. il che introduce un primo elemento di difficoltà, infatti non può essere definita una funzione di densità di probabilità ne tanto meno usare gli strumenti classici dell'analisi dei dati come per esempio il predittore a k passi di un processo stocastico.
Di conseguenza essendo che la dinamica dell'ambiente $p(s', r \mid s, a)$ è sconosciuta tutti i metodi di programmazione dinamica vengono esclusi in automatico, lasciando la scelta tra i metodi MC e i metodi TD.
Di fatto poi nell'implementazione sono stati usati i metodi Temporal Difference, in particolare l'algoritmo Q-LEARNING, perché questi all'atto pratico si sono dimostrati più veloci nel risolvere compiti stocastici rispetto ai metodi Monte Carlo.
I parametri principali dell'algoritmo sono alfa, gamma, epsilon e window_size.
Per epsilon si è scelto un valore di 0.1 in quanto la politica deve avvicinarsi ad una politica avida, valori maggiori avrebbero fatto perdere questa condizione e l'algoritmo durante il suo funzionamento avrebbe testato quindi più volte un'azione diversa da quella ottimale rallentando il funzionamento dello stesso.
Il parametro gamma, ovvero il tasso di sconto, è stato impostato a 0.6 in modo tale che potesse tenere sufficientemente conto delle ricompense future.
Per alfa, il parametro di aggiornamento dell'algoritmo è stato invece scelto un valore di 0.1 perché è quello che ha consentito di ottenere i migliori risultati, infatti un valore basso significa che i dati in tabella vengono aggiornati più lentamente rispetto a quando si usa un valore alto e questo è stato fatto per evitare che un casuale movimento di mercato molto marcato potesse influire nella fase di allenamento.
L'ultimo parametro è window_size e rappresenta la dimensione dell'episodio, questo è stato cambiato più volte nelle diverse versioni per vedere se i risultati cambiassero e quello che si è notato è che più il numero di step di un episodio è il numero di episodi stesso che incide maggiormente sul risultato finale.
Infatti quando si è deciso di adottare la proporzione 15:1 (nella versione 2) ovvero 15 episodi per l'allenamento e uno per la valutazione l'andamento del capitale è stato

molto caotico come si può vedere in figura 4, questo sta a dire che l'agente non è stato allenato adeguatamente.
Per quanto riguarda la funzione di ricompensa, per evitare che l'agente vada fuori obiettivo, si è semplicemente scelto la differenza di prezzo tra l'entrata e l'uscita di una posizione, infatti questa differenza rispecchia poi l'effettivo guadagno o perdita dell'agente i quali sono direttamente correlati al suo obiettivo ovvero la massimizzazione del profitto.
L'idea base di questa scelta è quella di mantenere la semplicità, infatti nei giochi atari RL ha avuto successo anche perché la funzione di ricompensa è di facile definizione infatti è quella che massimizza il punteggio di una partita la quale implica quindi la vittoria non permettendo così all'agente di trovare vie alternative per massimizzare la ricompensa attesa senza raggiungere gli obiettivi.
Un altro punto da discutere riguarda il non utilizzo di indicatori tecnici.
Come si può vedere da internet ci sono molte implementazioni di RL applicate al mondo del trading che addestrano l'agente mediante anche l'uso di indicatori tecnici ma così facendo si impone un determinato comportamento da seguire in funzione dei valori di queste funzioni di trading, il quale è contro l'idea base su cui si basa RL ovvero imparare dagli errori e lasciar libero l'agente di scegliere la strada che meglio ritiene per raggiungere l'obiettivo prefissato.
Infine i dati su cui è stato allenato l'agente sono quelli dello strumento finanziario NASDAQ 100, questa scelta è stata fatta perché questo asset è un indice, ovvero un paniere di azioni, il che gli attribuisce un andamento più lineare e meno variabile rispetto ad altri facilitando così l'agente nella costruzione della politica.

ANALISI DEI RISULTATI OTTENUTI

A prima vista i risultati ottenuti sull'anno 2020 (figura 1) e sull'anno 2021 (figura 2) possono sembrare molto soddisfacenti ma questi nascondono una pecca ovvero che sono dipendenti dai dati, in particolare dal trend dello strumento finanziario.
Infatti seppur l'agente ha seguito una fase di allenamento e valutazione su un set di dati differente, per avere dei buoni risultati finali questi set devono avere entrambi un andamento rialzista o ribassista, questo perché la politica imparata sotto una condizione (es. trend rialzista) chiaramente non può essere generalizzata all'altra condizione (trend ribassista).
Per ovviare a questo problema si è pensato di riproporzionare in modi differenti le fasi di allenamento e quella di valutazione, ovvero, piuttosto di 2 anni di allenamento e 1 di valutazione si è cercato all'interno dell'anno di susseguire x fasi di allenamento a y

fasi di valutazione, in particolare la combinazione che ha dato risultati migliori è stata la 15:1 ma in ogni caso non soddisfacente.
L'idea che sta alla base di questa scelta è quella di cercare di addestrare l'agente in entrambe le condizioni, infatti anche se il trend di un anno del NASDAQ 100 è positivo questo è composto sia da fasi rialziste che ribassiste di dimensione temporale più piccole, il problema è che una fase di allenamento può cadere a ridosso di entrambe le sotto fasi e quindi non dando una direzione ben definita alla politica (oltre che ad allenare l'agente con un numero di episodi esiguo), questo spiega l'andamento caotico della figura 4.
In conclusione si può dire che il reinforcement learning incontra diverse problematiche quando applicato al mondo del trading le quali possono essere riassunte in:

- Ambiente che varia troppo rapidamente non lasciando quindi il tempo all'agente per adattarsi.
- La politica che un'agente impara in un periodo potrebbe non essere più valida per il periodo successivo, quindi anche l'utilità di un'applicazione del genere viene a meno perché non può essere generalizzata ad un'altra situazione.
- L'ambiente in cui è fatto operare l'agente è solo parzialmente osservabile, infatti le sue scelte si basano su dati di mercato ma quest'ultimi dipendono dall'andamento dell'economia, finanza ecc. ovvero tutti fattori che influiscono poi nelle scelte degli investitori e che l'agente non tiene in considerazione.

Analizzati questi punti si è concluso che forse piuttosto che usare RL nel mondo del trading con lo scopo di massimizzare il profitto potrebbe essere invece usato per aiutare il trader a impostare i parametri di indicatori tecnici dai quali poi estrarre segnali per eventuali aperture di posizioni.
Per esempio potrebbe essere applicato ad una media mobile a n periodi per cercare di ottenere il valore di n che massimizza le vincite.