**Primo assignment – Multi Arm Bandints**

Nel problema del 10-Arm Bandit abbiamo n differenti slot machine a cui giocare. Possiamo scegliere di giocare ad un solo slot alla volta e, una volta giocato, otteniamo una ricompensa. Il nostro obiettivo è quello di massimizzare le ricompense cumulative ottenute giocando agli slot.

Assumiamo che l’insieme delle nostre azioni è l’insieme di tutte le slot machine e al tempo t scegliamo a quale slot giocare. Una volta giocato, otteniamo una ricompensa scelta da una distribuzione di probabilità stazionare che dipende dall’azione e definiamo il valore di una azione come la ricompensa attesa data l’azione selezionata. Tuttavia, questa quantità non è nota, ma conosciamo solamente una sua stima.

La stima del valore di una azione viene calcolata tramite la media delle ricompense ottenute al passo t-esimo.

Nella nostra applicazione di questo problema assumiamo che il valore delle azioni siano inizializzate allo stato valore e che aumentano e diminuiscono indipendentemente una dall’altra in ogni istante di tempo rendendo il problema non stazionario. Inoltre, abbiamo confrontato i vari metodi con vari valori di ϵ e α- Quindi, quello che dovevamo fare era confrontare i vari metodi azione-valore e abbiamo scelto di mettere un seed per rendere confrontabili tutti i vari metodi.

Il primo è quello ϵ-greedy per ambienti stazionari, impostando il passo di aggiornamento come α=1/k e come si può evincere dal grafico nel caso di sample avarege con passo di aggiornamento decrescente con ai valori valori di epsilon manteniamo l’esplorazione consentendoci di scegliere azioni che non sono epsilon greedy e più o meno in 1000 istanti di tempo convergono quasi tutte al valore vero. Tuttavia, questa soluzione non è propriamente corretta poiché il passo di aggiornamento non è costante. In basso, abbiamo graficato per ogni istante di tempo quando scegliamo l’azione ottima e quando no.

Nel secondo metodo action-value con passo di aggiornamento costante nel tempo, notiamo che la ricompensa cumulativa oscilla attorno al valore vero e vi è maggiore varianza e per vari valori di ϵ notiamo che man mano che cresce tendiamo a non prendere sempre l’azione ottimale.

Nel terzo metodo action-value detto Upper Confidence Bound notiamo che per vali valori di c, che rappresenta il grado di esplorazione, notiamo che le azioni vengono scelte più o meno tutte equamente e il termine sotto radice ci garantisce la scelta indefinita di tutte la azioni nel tempo.

Infine, abbiamo il metodo basato sulle preferenze, notiamo che prediligiamo la prima azione ed inoltre, come dice l’algoritmo, tutte quelle volte in cui una azione non viene scelta la preferenza di esser scelta diminuisce avendo così anche valori negativi. Il valore per cui si avvicina di più è per alpha pari a 0.01.

**Secondo Assignment – Tic Tac Toe**

L’assignment del Tic Tac Toe consisteva nell’applicare le tecniche di Policy Iteration e Value Iteration nel gioco del Tic Tac Toe. In particolare, occorreva determinare la matrice di transizione che rappresenta le propabilità di giungere in uno stato s’, partendo dall’azione s e selezionando l’azione a e la matrice delle ricompense che contiene le ricompense per ogni che si ottengono partendo da uno stato s selezionando lo stato a. A tale scopo, dovevamo costruire l’ambiente del Tic Tac Toe considerando che la macchina che deve apprendere gioca per primo, che si considerano solamente afterstates cioè configurazioni della board in cui sono presenti in egual numero segni della croce e del cerchio.

Sotto queste ipotesi per ottenere le board valide abbiamo effettuate considerazioni sulle configurazioni delle board. Prima ci siamo calcolati tutte le possibili combinazioni dei segni in una board eliminando quelle che erano composte da tutte X/O e quelle che avevano un numero troppo elevato di segni X/O e quelle in cui gioca per prima il cerchio, cioè l’avversario che gioca casualmente. In seguito, abbiamo notato che le board potevano essere ruotate in 8 modi differenti e quindi, levando anche queste board, siamo giunti a 341 stati. In cui le ultimi tre stati rappresentano tutte le board terminali in cui si vince, perde o pareggia. A questo punto per determinare le board successive abbiamo prima ordinato le board per numero crescente di segni 2,4,6,8 e, procedendo per differenza tra board, con la condizione che una board è successiva se e solo se negli spazi rimanenti vi siano solamente un cerchio e una croce, abbiamo cercato nelle varie partizioni che abbiamo creato gli stati successi a cui abbiamo associato l’azione che mi porta da s in s’. La struttura risultante è simile ad un albero con la possibilità di scendere solamente in profondità sia nelle rotazioni che negli stati successivi.

Quindi, una volta ottenuta la tripla s,a,s’ abbiamo assegnato la probabiltà come 1/|a| cioè la cardinalità delle azioni. Una volta riempita la matrice di transizione abbiamo controllato che la somma di ogni riga sia pari ad 1 poiché si tratta di probabilità. Dato che abbiamo considerato solamente gli stati unici, nel riempire la matrice di transizione, abbiamo tenuto particolare attenzione all’includere anche le rotazioni delle board. Infine per quanto riguarda la matrice delle ricompense abbiamo assegnato reward pari a 0 per ogni coppia s,a che mi porta in uno stato non terminale e +1/-1 rispettivamente per vincita e sconfitta/pareggio.

A questo punto, abbiamo utilizzato il policy iteration e value iteration per ottenere la policy ottima che partendo da un qualsiasi stato mi dicesse l’azione ottima che mi massimizza la ricompensa. Dato che abbiamo considerato solamente stati unici, per rendere giocabile Tic Tac Toe con qualsiasi rotazione delle board, abbiamo creato una funzione che antiruotasse le board in quelle principali che abbiamo considerato nella costruzione delle matrici delle ricompense e di transizioni per avere un mapping tra la policy ottima e quella su quale stiamo effettivamente giocando.

Infine, abbiamo notato che fra il PI e VI, l’algoritmo che ci garantiva più vittorie è il PI e molto spesso la nostra intelligenza artificiale preferiva fare il doppio gioco piuttosto che vincere subito. Questo poiché si assicura una vittoria certa. In un solo caso, si ha il “triplo gioco” in cui l’agente poteva vincere in 3 modi differenti.

**Terzo Assignment – Race Car**

Il terzo Assignment Race Car consiste nell’applicare il metodo di Monte Carlo per addestrare una macchina a compire una curva a massima velocità senza uscire fuori la curva. In particolare, la macchina si trova in una posizione discreta della curva, la sua velocità deve essere compresa fra 0 e 5.

L’agente può effettuare 2 azioni per incrementare le componenti della velocità orizzontalmente e verticalmente. Ad ogni istante di tempo si assegna reward -1.

Ogni episodio parte casualmente dalla linea di partenza e termina alla linea di fine e nel caso in cui si esce dal tracciato l’episodio ricomincia dalla linea di partenza.

L’ambiente utilizzato è il GridWorld di matlab a cui abbiamo inserito due parabole come ostacoli per rappresentare la curva. In particolare, ogni punto della parabola è selezionato in un intorno casuale per creare delle gole ed avere il tracciato non costante. Inoltre, dato che l’agente può incrementare contemporaneamente entrambe le componenti delle velocità abbiamo creato una funzione che, proiettando il cammino che avrebbe dovuto fare la macchina, ci permettesse di rilevare gli ostacoli e quindi limitare le azioni disponibili all’agente.

Inoltre, abbiamo saturato sia la posizione che la velocità della macchina per non farla uscire dal tracciato. La scelta dell’azione è effettuata in maniera epsilon greedy.

Grazie ai metodi di Monte Carlo una volta terminato l’episodio si calcola il ritorno e si propagano all’indietro tutte le informazioni ottenute permettendo di aggiornare la policy con una migliore.

**Quarto Assignment – Minimum Time Lift**

Per la modellazione dell’ambiente abbiamo scelto di scrivere la dinamica dell’ascensore, che per semplicità l’abbiamo posta pari ad 1, in x1 e x2. Questa coppia rappresenta lo stato x in cui la x1 è la posizione dell’ascensore e x2 è la velocità. Le corrispettive derivate delle componenti dello stato ci forniscono rispettivamente velocità e accelerazione. Proprio quest’ultima è il controllo che il nostro agente andrà ad attuare. L’agente sceglie di accelerare/decelerare al massimo o procedere di moto rettilineo uniforme. Dato che lo spazio di stato ed in teoria anche le azioni sono continue, per calcolare lo stato s’ andiamo a risolvere in maniera discretizzata il sistema della dinamica xdot = Ad\*x+B\*u. La discretizzazione del sistema l’abbiamo presa dall’ambiente di matlab del doppio inegratore. Inoltre, matlab ci consentiva di calcolare l’indice di costo ottimo tramite x.’Q\*x+u.’R\*u, tuttavia noi abbiamo deciso di assegnare reward -1 per ogni istante di tempo. Nel caso in cui l’ascensore arriva al lower e upper bound della posizione abbiamo deciso di saturare la posizione e di ipotizzare che la velocità sia nulla, mentre l’accelerazione no. Quindi è come se l’ascensore stesse provando ad accelerare e di procedere in moto uniformemente accelerato, ma vi è un vincolo che può essere rappresentato sia dalle pareti in cui risiede l’ascensore.Data la numerosità degli stati, abbiamo utilizzato gli approssimatori funzionali e il tile coding. In particolare, abbiamo utilizzato 4 griglie di grandezza 20, 2 feature rappresentanti la velocità e un offset di (1,3). Le azioni vengono prese in maniera greedy.

Tramite il tile coding lineare andiamo a creare 4 griglie con celle non sovrapposte fra di loro. Nell’intersezione di ogni griglia vi è una feature.

**Metodo di Monte Carlo**

I metodi di Monte Carlo sono metodi di apprendimento per task episodici che non assumono la conoscenza dell’ambiente e richiedono solamente esperienza ottenuta simulando il modello. La parte di predizione dei metodi di monte carlo riguarda la stima della policy tramite le visite dello stato negli episodi. Si divide in first-visti e every-visit: nel primo si stima il valore della policy solo nella prima visita nello stato, nel secondo ogni qual volta si visita lo stato nell’episodio. Il secondo metodo converge quadraticamente.

Nel controllo dei Metodi di Monte Carlo si utilizza l’idea del GPI, cioè si osserva l’intero episodio alternando fasi di improvement e di evaluation.

I metodi più efficaci sono i metodi di Monte Carlo off-policy che valutano o migliorano la policy solo dopo che si sia visto tutto l’episodio, ma la convergenza è anche più lenta. In questi metodi si introduce un fattore di importanza che permette di valutare il valore atteso di una policy avendone già vista un’altra. Ciò consente di ridurre il bias ed avere una varianza limitata.

**Temporal Difference**

A differenza dei metodi di Monte Carlo in cui si aspetta fino alla fine dell’episodio per osservare il ritorno, i metodi alle differenze temporali utilizzano il ritorno istantaneo o a n-passi (visione all’indietro o in avanti).

In questi metodi si utilizza l’errore delta per muovere il valore della funzione valore di stato verso una stima migliore e quindi occorre aspettare fino all’istante di tempo successivo.

I vantaggi sono che può essere implementato online, richiedono un singolo passo, l’episodio può anche non terminare e può essere applicato sia per task continui che episodici.

Per quanto riguarda il target, a differenza di Monte Carlo in cui il ritorno è affetto da bias, il target vero del TD non è affetto, ma quello istantaneo si.

**Monte Carlo vs Temporal Difference**

I metodi di Monte Carlo hanno alta varianza e bias nullo:

* Buone proprietà di convergenza;
* Non è molto sensibile al valore iniziale;
* Facile da capire e usare;
* Più efficiente in ambienti non markoviani;

I metodi TD hanno bassa varianza e bias non nullo:

* Di solito sono più efficienti dei metodi di Monte Carlo;
* TD(0) converge al valore vero della funzione valore;
* Sensitivo ai valori iniziale;
* Più efficiente in ambienti Markoviani.

**SARSA**

Negli algoritmi di tipo SARSA si utilizza il temporal difference TD(‘) per stimare i valori delle azioni e l’aggiornamento è effettuato in base alla quintupla Stato-Azione-Reward-Stato-Azione e convergono con probabilità 1. Vi sono tante implementazioni alternative:

* **Expected SARSA**: si considera l’aggiornamento del Q-Learning *(Il metodo Q – Learning è indipendente dalla policy che si sta seguendo e cerca di approssimare direttamente la funzione qualità ottima. Non si richiede importance sampling.In particolare, la prossima azione è scelta usando una policy comportamentale e si considera un’azione alternativa seguita dal target della policy greedy.In questo tipo di algoritmo sia il comportamento che le policy del target migliorano.Q – Learning converge con probabilità 1.)* Questo consente di eliminare la varianza dovuta alla selezione casuale dell’azione all’istante successivo.
* **N-Step SARSA:** i ritorni ad n passi possono essere suddivisi in termini di valore di azioni, le azioni sono scelte secondo una politica epsilon greedy rispetto alla funzione qualità e, utilizzando l’importance sampling, si ottengono algoritmi off-policy.
* **SARSA(λ):** Il SARSA(λ) è un metodo di controllo on-policy che utilizza la predizione del TD(λ) per le coppie stato azione e l’aggiornamento della stima della funzione qualità dipende sia dall’errore delle differenze temporali sia dalle tracce di elegibilità. Le tracce vengono annullate ogni qual volta si prende un’azione esplorativa.

**Tracce di Eleggibilità**

Le tracce di elegibilità forniscono una memoria addizionale per ogni stato. Ad ogni istante, le tracce di elegibilità non visitate decadono mentre quella selezionata viene aumentata di 1. Quindi le tracce di elegibilità indicano il grado di elegibilità di un evento di apprendimento. In TD(λ) l’errore alle differenze temporali è proporzionare alle tracce di elegibilità. Ogni volta che lo stato viene visitato, si aggiorna la traccia di elegibilità, in alternativa viene decrementata.

Queste tracce possono essere aggiornate in 3 modo:

* Accumulating: ogni volta che si visita uno stato, la sua relativa traccia viene incrementata di 1, il problema è che la traccia può diventare maggiore di 1;
* Replacing: ogni volta che uno stato viene visitato, la traccia viene impostata a 1;
* Dutch: è un aggiornamento intermedio tra l’accumulating e il replacing. In base al valore di alpha si ha un accumulating o un replacing.

**Q – Learning**

Il metodo Q – Learning è indipendente dalla policy che si sta seguendo e cerca di approssimare direttamente la funzione di qualità ottima. Non si richiede importance sampling. La prossima azione viene scelta usando una policy comportamentale e si considera un’azione alternativa seguita dal target della policy greedy. Converge con probabilità 1.

**Dyna – Q**

L’algoritmo Dyna – Q, utilizzati per apprendere un modello, utilizza un modello di apprendimento deterministic con il metodo Q – Learning ad un passo. In particolare, si campiona casualmente le coppie stato azioni che sono state visitare, il modello ritorna come predizione l’ultimo stato e le ricompense successive osservate. La sua versione Q+ tiene traccia per ogni coppia stato azione di quanto tempo è trascorso dall’ultima visita. Più tempo trascorre, più il modello non è correto.

**Double Learning**.

Il Double Learning è un TD control modificato in cui si dividono i dati in due insiemi e si utilizzano entrambi per apprendere due indipendenti stime della funzione valore. La prima stima viene utilizzata per determinare l’azione massimizzante, la seconda per stimare il suo valore eliminando così il bias.