Discussione Progetto

Il Blackjack è un popolare gioco di carte giocato in molti casinò. L'obiettivo del gioco è quello di vincere denaro ottenendo un punteggio totale superiore a quello del banco senza superare 21. Le regole classiche del gioco sono state cambiate in modo da semplificare il problema in questo modo:

1. La partita si gioca con un mazzo di carte infinito.
2. Ogni estrazione della carta fornisce come risultato un valore compreso tra 1 e 10 con una la probabilità che esca una carta di colore rosso(1/3) o una nera (2/3)
3. Gli assi possono assumere solo il valore 1 e non 11 come nel gioco originale.
4. All'inizio del gioco sia il giocatore e il banco pescano una carta nera.
5. Ad ogni turno il giocatore può chiedere una carta o fermarsi.
6. I valori delle carte dei giocatori sono aggiunti alla somma totale se la carta estratta è nera o sottratta se è rossa
7. Se la somma giocatori è superiore a 21, o diventa inferiore a 1, il giocatore perde la partita (reward -1).
8. Se il giocatore decide di fermarsi allora comincia il turno del banco. Il banco decide sempre di fermarsi quando la somma delle sue carte è uguale o superiore di 17. Se la somma del banco è superiore a 21 o inferiore a 1, allora il giocatore vince. Altrimenti si vede chi dei due ha la somma delle carte più alta e la partita può finire con una vittoria del giocatore (reward +1), sconfitta (reward -1) e infine pareggio (reward 0).

Il problema è stato affrontato come un MDP <S,A,T,R> dove non c'è discounting $\gamma = 1$. Ricordiamo che la costante γ(con$0 \leq \gamma < 1$) è detto *fattore dis conto* e rappresenta l'interesse dell'agente rispetto alle ricompense che può ricevere "*in futuro*":

– Se γ è vicino ad 1 allora le ricompense future sono rilevanti per l'agente.

– Se γ è vicino ad 0 allora le ricompense future poco rilevanti per l'agente (per $\gamma = 0$,viene considerata solo la ricompensa immediata).

- States: (P,D) where
  – P: current sum of player card values.
  – D: current sum of dealer (when is at least 17, the dealer stop to HIT)

- Action: HIT, STICK (the player HIT when he wants another card, stay if he is satisfied of the value of the sum)

- Transition function: where c is the value of new card extracted.
  Table 1: Transition table State Action Transition
  (P,D) STICK (P,D) , (P,D) HIT ($P \cup c$, D)

• Reward: If the dealer goes bust, then the player wins; otherwise, the outcome win (reward +1), lose (reward -1), or draw (reward 0) is the player with the largest sum.

Gli algoritmi scelti per la risoluzione di tale problema sono stati: Monte Carlo method e Sarsa(λ)

MONTE CARLO

A differenza della programmazione dinamica, non è necessario conoscere la probabilità di tutte le transizioni possibili. In molti casi infatti è semplice generare dei campioni che soddisfino le distribuzioni di probabilità desiderate, mentre è impraticabile esprimere in modo esplicito la totalità delle distribuzioni di probabilità.

Questi algoritmi simulano una sequenza di esempio, chiamata episodio, e in base ai valori osservati aggiornano le stime delle value function e le policy. Iterando per un numero sufficiente di episodi i risultati ottenuti mostrano un accuratezza soddisfacente.

Rispetto agli algoritmi della programmazione dinamica, gli algoritmi Monte Carlo non necessitano del modello completo del sistema. Tuttavia, offrono la possibilità di aggiornare la value function e politicy solamente al termine di ogni simulazione, a differenza degli algoritmi di DP che aggiornano le stime ad ogni passo.

In particolare, supponiamo di voler stimare V π (s), il valore di uno stato s sotto π politica, dato un insieme di episodi ottenuti seguendo π.

Pertanto la seguente equazione mostra l'aggiornamento della Value Function:

V (st) ← V (st) + α [Rt -V (st)] (1) dove Rt è il reward effettivo a lungo termine seguendo lo stato St.

In particolare i metodi Monte Carlo hanno questa sorta di ciclo nel trovare la policy ottima che li conduce alla value function ottima e viceversa. Ciò avviene tramite le azioni di evaluation e improvement. Nella evaluation la value function è ripetutamente aggiornata in base alla policy corrente, mentre nell improvement la policy è ripetutamente migliorata in base alla corrente value function.

TD

Gli algoritmi di temporal difference learning (TD learning), nascono come combinazione di idee prese dalla programmazione dinamica e i metodi Monte Carlo. Come i metodi Monte Carlo, possono apprendere direttamente da degli episodi di esempio, senza un modello completo della dinamica del sistema. Come gli algoritmi di DP, aggiornano le stime basandosi in parte su altre stime apprese precedentemente, senza dover per forza attendere la fine dell'episodio.

Il metodo più semplice TD, noto come TD (0), è:

V (st) ← V (st) + α [rt + 1 + γV (st + 1) -V (st)] (5)

In effetti, per l'aggiornamento per il reward in Monte Carlo è Rt, mentre in TD è rt + 1 + γV (st + 1).

I metodi basati sulle differenze temporali permettono di gestire il problema del controllo (ovvero di ricerca della policy ottima) attraverso l'aggiornamento delle "value function" in base agli esiti della transizione allo stato successivo. Ad ogni passo la funzione Q (action-value function) viene aggiornata sulla base del valore da essa assunto per la coppia stato-azione successiva e dalla ricompensa ottenuta attraverso la relazione:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

in cui α rappresenta un parametro costante legato all'apprendimento e γ è il fattore di sconto.

Un aspetto caratterizzante i diversi tipi di algoritmi basati su differenze temporali è la metodologia di scelta di un'azione. Esistono metodi alle differenze temporali "on-policy", in cui l'aggiornamento

viene effettuato sulla base dei risultati di azioni determinate dalla policy selezionata e metodi “off-policy”, in cui diverse policy possono essere valutate tramite azioni ipotetiche, non effettivamente intraprese. A differenza dei metodi “on-policy”, questi ultimi possono separare il problema dell’esplorazione da quello del controllo, imparando tattiche non necessariamente applicate durante la fase di apprendimento.

Algoritmo SARSA e Sarsa(λ)

L’algoritmo SARSA implementa un metodo alle differenze temporali “on-policy”, in cui l’aggiornamento della funzione Q (action-value function) viene effettuato in base agli esiti della transizione dallo stato s=st allo stato s’=st+1 tramite l’azione at , intrapresa in base ad una policy selezionata π(s, a).

Esistono policy che scelgono sempre l’azione che fornisce la massima ricompensa e policy non-deterministiche (ε-greedy), che assicurano un elemento di esplorazione nella fase di apprendimento come nel mio caso.

Tale algoritmo combinato con il concetto di Eligibility trace da vita a Sarsa(λ)

L’ Eligibility trace traccia di eventi passati o futuri in base al fatto se è forward o backward e quando viene calcolato un errore usando metodi basati su TD, la eligibility trace suggerisce quali variabili aggiornare.

In definitiva amplia l’orizzonte temporale sul quale fare l’aggiornamento a più di 1 passo.

L’aggiornamento del Q value quindi sarà:

$$Q_{k+1}(s_t,a_t)=Q_k(s_t,a_t)+\alpha\left[r_{t+1}+\gamma Q_k\left(s_{t+1},a_{t+1}\right)-Q_k\left(s_t,a_t\right)\right]$$

Errore: δ_t

$$Q_{k+1}(s_t,a_t)=Q_k(s_t,a_t)+\alpha\delta_t$$ Per 1 coppia (s,a)

$$Q_{k+1}(s,a)=Q_k(s,a)+\alpha\delta_t e_t(s,a)$$ Per tutte le coppie (s,a)

Eleggibilità: $e_t(s,a)$

Dove

$$e_t(s,a)=\begin{cases}\gamma\lambda e_{t-1}(s,a)+1 & \text{if } s=s_t \text{ and } a=a_t;\\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise.}\end{cases}$$