**Ricompensa**

La ricompensa è un segnale di feedback scalare ed indica come si comporta l’agente al tempo t. Rappresenta la quantità da massimizzare.

**Agente**

L’agente è colui che effettua le azioni. Riceve le osservazioni e le ricompense dall’ambiente.

**Ambiente**

L’ambiente è tutto ciò che circonda l’agente. Riceve le azioni dell’agente da cui genere le osservazioni e le ricompense.

**Policy**

La policy è la regola utilizzata per prendere le azioni. Può essere deterministica o stocastica

**Funzione valore**

La funzione valore è la stima di quanto buona è una azione in ogni stato e predice la ricompensa.

**Modello**

Il modello è la credenza dell’agente sul comportamento dell’ambiente.

**Predizione**

La predizione valuta il futuro data una policy.

**Controllo**

Il controllo è l’ottimizzazione del futuro data la stima della policy.

**Bootstrapping**

Il boostrapping è una tecnica utilizzata in alcuni metodi di apprendimento come la programmazione dinamica in cui si utilizza una certa stima per stimare il valore della funzione valore.

**Multi-Arm bandit**

Il problema del multi arm bandit è un problema non associativo. In questo problema dobbiamo ripetutamente scegliere tra diverse azioni. Dopo ogni scelta si riceve un reward numerico scelto da una distribuzione di probabilità stazionaria che dipende dall’azione selezionata e l’obiettivo è quello di massimizzare la ricompensa attesa totale dopo un certo periodo di tempo.

Il valore di una azione dipende dalla ricompensa attesa data l’azione selezionata.

**Multi-ArmBandit – Esplorazione vs Sfruttamento**

Un’azione è detta greedy se scelta dal più alto valore della stima della funzione valore e corrisponde allo sfruttamento della conoscenza ottenuta. Altrimenti, si esplora.

Quindi, lo sfruttamento punta a massimizzare la ricompensa in un passo mentre l’esplorazione produce una ricompensa maggiore a lungo termine. Il compromesso è quello di bilanciare questi due aspetti: se la varianza della ricompensa è grande, necessitiamo una maggiore esplorazione; altrimenti sfruttiamo.

**Multi-ArmBandit - Stima funzione valore**

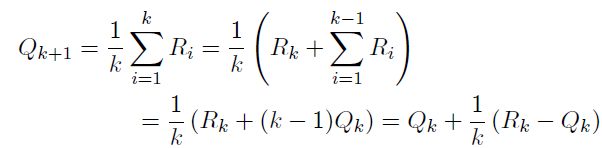
Al tempo t, una stima della funzione valore può essere determinata tramite la media delle ricompense ottenute per quella determinata azione e spesso questa stima converge al valore vero all’infinito.

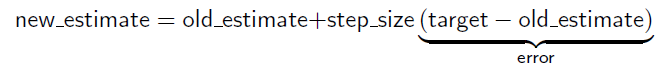
**Multi-ArmBandit – metodo epsilon greedy**

La selezione di un’azione greedy sfrutta la conoscenza corrente per massimizzare la ricompensa istantanea. Tramite i metodi epsilon greedy si selezione con probabilità epsilon una azione casuale fra tutte quelle disponibili.

**Multi-ArmBandit – Implementazione iterativa**

La stima della funzione valore può essere calcolata iterativamente:





La scelta di un passo di aggiornamento pari a 1/k garantisce la convergenza stocastica, cioè: Immagine che contiene testo

Descrizione generata automaticamente

**Multi-ArmBandit – Problemi non stazionari**

Se l’ambiente è non stazionario, cioè il valore vero della funzione valore cambia nel tempo, si preferisce utilizzare un passo di aggiornamento costante alpha, compresa fra 0 ed 1, invece di utilizzare 1/k. La conseguenza di questa scelta è che le ricompense hanno un minor impatto.

**Multi-ArmBandit - Inizializzazione**

Fino ad ora, la stima della funzione valore dipende dalla stima iniziale e il bias dovuto a questa scelta diminuisce nel tempo. Tuttavia, è possibile utilizzare questa conoscenza per promuovere l’esplorazione: valori grandi della stima iniziale rispetto alla ricompensa attesa promuovono l’esplorazione anche sotto la condizione di azioni greedy. Quindi si provano numerose azioni prima di avere la convergenza al valore vero della funzione valore.

**Multi-ArmBanditi – UCB Upper Confidence Bound**

L’idea dell’upped confidence bound è quella di selezionare tra tutte le azioni non greedy quella azioni che potenzialmente potrebbe essere ottimale:

Immagine che contiene testo

Descrizione generata automaticamente

Il parametro c controlla il grado dell’esplorazione mentre il termine sotto radice rappresenta la misura dell’incertezza. Il tempo t aumenta in ogni istante di tempo permettendo la selezione di ogni azione indefinitamente nel tempo.

**Multi-Arm Bandit – Preference Update**

Sia Ht(a) la preferenza della selezione dell’azione a. Le azioni vengono quindi selezionate basandosi su una distribuzione soft-max ed inizialmente tutte le preferenze sono poste allo stesso valore.

Dopo aver selezionato l’azione At e ricevendo la ricompensa Rt, le preferenze vengono aggiornate nel seguente modo:

Immagine che contiene testo

Descrizione generata automaticamente

La media Rt funge da lower bound del reward. Nel caos in cui la ricompensa all’istante t è più grande della baseline, la probabilità di selezionare quella determinata azione aumenta; altrimenti diminuisce. Durante l’esecuzione di questo algoritmo, la preferenza delle azioni non selezionati diminuisce.

Infine, questo algoritmo è un’applicazione del gradiente stocastico ascendente.

**Markov Decision Process**

**Proprietà di Markov**

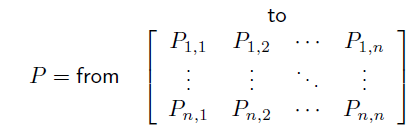
Il futuro è indipendente dal passato dato il presente. Quindi lo stato contiene tutte le informazioni rilevanti della storia e, una volta noto, tutta la storia può essere non considerata.

**Matrice di transizione dello stato**

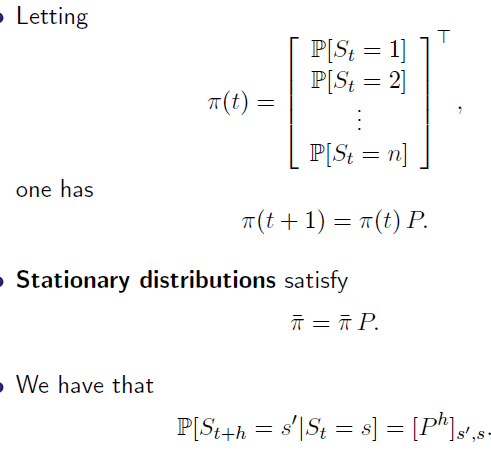
Dato uno stato e il suo successivo è possibile definire la probabilità di transizione:



Se gli stati sono finiti è possibile definire la matrice di transizione in cui la somma di ogni riga è pari ad 1 (poiché si tratta di probabilità)



**Distribuzione di probabilità nelle catene di Markov**

****

**Processo di Markov**

Un processo di Markov è un processo casuale senza memoria di una sequenza di stati che soddisfanno la proprietà di Markov.

**Catena di Markov**

Una catena di Markov è una coppia (S,P) con S insieme finito di stati e P matrice di transizione.

**Interazioni tra agente e ambiente in un processo markoviano**

Ad ogni istante di tempo l’agente osserva lo stato dell’ambiente, seleziona un’azione A, riceve un reward R e decide e ritrova da solo nel nuovo stato. La dinamica viene definita dalla probabilità di trovarsi nello stato s’ con reward r dato lo stato e l’azione:

Immagine che contiene testo

Descrizione generata automaticamente

**MDP – Funzioni di propabilità**

**Immagine che contiene testo

Descrizione generata automaticamente**

**Task Episodici – Ritorni**

Le interazioni tra l’agente e l’ambiente possono essere suddivisi in sotto-sequenze dette episodi. Il tempo di terminazione di un episodio è differente da episodio e episodio.

Inoltre, ogni episodio finisce un uno stato speciale detto stato terminale che è seguito da un reset. Date queste assunzioni, possiamo definire il ritorno atteso come la somma delle ricompense ottenute nell’episodio.

**Task Continuativi – Ritorni**

Se trattiamo task continuativi, dobbiamo introdurre un fattore di sconto gamma compreso fra 0 e 1 e il ritorno atteso viene definito come:

Immagine che contiene testo

Descrizione generata automaticamente

Il parametro gamma se tende a 0 si effettua una valutazione greedy e permette di non avere infinite ricompene se R è limitato; se tende ad 1 si effettua una valutazione a lungo termine.

Per definizione in uno stato terminale la ricompensa è nulla.

**MDP – Policy**

Una policy è un mapping dallo stato corrente alle probabilità di selezionare un’azione a ammissibile.

**MDP – Funzione Valore State**

La funzione valore di uno stato s sotto una policy pi è il ritorno atteso partendo da s e seguento la policy p: 

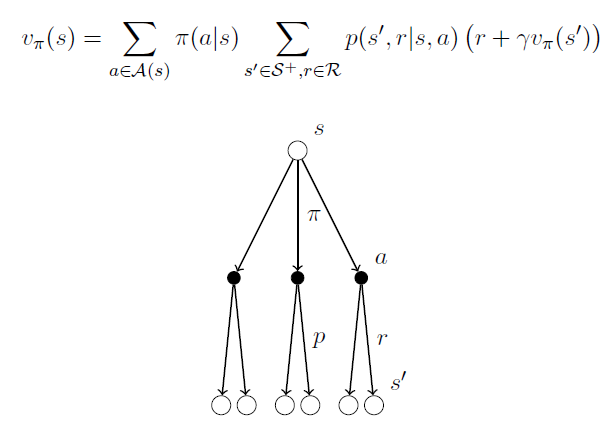
**MDP – Funzione valore stato azione**

La funzione valore stato-azione è anche detta funzione qualità ed è il ritonro atteso partendo da s, prendendo l’azione a, e continuando con la policy pi:



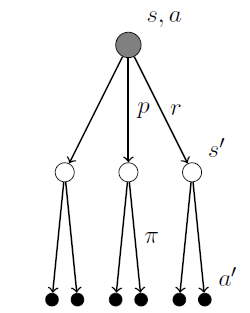
**MDP – Equazione di Bellman e Diagramma di Backup**

**Immagine che contiene testo

Descrizione generata automaticamente**

**Immagine che contiene testo

Descrizione generata automaticamente**

****

La funzione valore viene utilizzata per oridnare le policy permettendo di determinare la bontà di una policy rispetto ad un’altra. La policy che ha un valore maggiore o uguale a tutte le altre viene detta policy ottimale. Tutte le policy con queste caratteristiche condividono la stessa funziona valore e la stessa funzione qualità ottenute effettuando il massimo tra tutte le funzioni valore/qualità trovate. In aggiunta, sia la funzione valore che qualità soddisfano l’equazioni di Bellman.

**MDP – Problematiche equazione di Bellman**

Le problematiche della risoluzione dell’equazione di Bellman è che la dinamica dell’ambiente non è sempre nota con certezza, la risoluzione dell’equazione può risultare costosa e non sempre gli stati soddisfano la proprietà di Markov.

**Programmazione Dinamica**

La programmazione dinamica è un metodo utilizzato per risolvere problemi complessi basato sulla suddivisione del problema originario in sotto-problemi le cui soluzioni sono combinati per ottenere la soluzione del problema originario.

La programmazione dinamica richiede la conoscenza della funzione p(s’r|s,a) e viene utilizzato per pianificare la policy. Inoltre, stime della p(s’,r|s,a) sono utilizzate per apprendere. Quindi:

* Si ha la predizione quando, data una policy p, si trova la funzione valore;
* Controllo date le funzioni valori, si trova la policy ottima.

**Policy Evaluation**

Il policy evaluation coinvolge il problema di predizione per cui data una policy, si vuole stimare la funzione valore definita tramite l’equazione di Bellman. L’unicità della soluzione viene garantita per gamma<1 e dalla terminazione di tutti gli stati secondo la policy.

**Iterative Policy Evaluation**

Nella versione iterativa del policy evalutation si applica iterativamente l’equazione di Bellman approssimando la funzione valore. Queste approssimazioni vengono dette backup. In particolare si inizializza la stima della funzione valore arbitrariamente per tutti gli stati tranne quelli terminali in cui è impostata a 0. In seguito, si utilizza l’equazione di Bellman come regola di aggiornamento, effettuando backup per tutti gli stati in maniera sincrona. Esiste una versione alternativa del policy evaluation in cui si aggiornano i valori della funzione valore ogni qual volta che è disponibile una stima e, solitamente, questa versione converge più velocemente.

**Policy Improvement**

Tramite il policy evalutation sappiamo quando è buona una policy pi. Per migliorare una determinata policy occorre utilizzare la funzione qualità che prende in considerazione la qualità di prendere un’azione rispetto ad un’altra per poi proseguire con la policy assegnata.

Grazie al **policy Improvement Thoerem** si garantisce che se la funzione qualità è maggiore della stima della funzione valore per un determinato stato, allora la funzione valore della nuova policy è maggiore di quella vecchia.

Il policy improvement agisce in maniera greedy rispetto la funzione qualità per migliorare una policy.

**Policy Iteration**

Nel policy iteration si alternano il policy evaluation e improvement per ottenere una sequenza monotona di funzioni valore crescente ottenendo una policy migliore.

Questo algoritmo converge in un numero finito di iterazioni. Tuttavia, si richiedono molti passi di policy evaluation e la convergenza è al limite.

**Value Iteration**

Il value iteration è una versione di policy iteration in cui si troncano gli aggiornamenti ad un singolo passo. Si basa sul concetto del **principio di ottimalità**: una policy ottima ha la proprietà che qualsiasi sia lo stato e le decisioni iniziali, le rimanenti decisioni devono costituire una policy ottima per quanto riguarda lo stato risultante dalla prima decisione.

Quindi, nel value iteration si unisce un passo di policy evaluation ed un passo di policy improvement. Le consuguenze di questa scelta sono che: nel passo di improvemente si rende la policy greedy rispetto alla funzione qualità, ma il valore della funzione valore non è corretta per la policy; mentre con il passo di evaluation si rende il valore della funzione valore consistente con la policy, ma tipicamente la policy non è più greedy.

**Efficienza della programmazione dinamica**

**Immagine che contiene tavolo

Descrizione generata automaticamente**

La programmazione dinamica soffre del problema del **curse of dimensionality:** ilnumero di stati cresce esponenzialmente con il numero di variabili portando ad una difficoltà computazionale non indifferente. A quel punto si preferisce la programmazione dinamica asincrona.

**Metodi 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.

A differenza del policy evaluation si utilizza il ritorno medio invece del ritorno atteso.

**Monte Carlo Prediction**

Data una policy, vogliamo determinare la sua stima dato un insieme di episodi ottenuti seguendo la policy pi attraverso s. Ogni ricorrenza dello stato s nell’episodio viene detta visita: i metodi di Monte Carlo first-visit stimano la funzione valore come la media dei ritorni seguendo la prima visita nello stato; i metodi di Monte Carlo every-visit stimano la funzione valore come la media dei ritorni seguendo tutte le visite nello stato s.

La convergenza di first-visit, il ritorno è indipendente e identicamente distribuito sulla funzione valore con una varianza finita. La convergenza every-visit converge quadraticamente.

Infine, la predizione dei Metodi di Monte Carlo non richiede la conoscenza delle probabilità, ma si considerano solamente le traiettorie in un episodio; non si effettua bootstrap e il costo computazionale è indipendente dal numero di stati.

**Stima Monte Carlo del valore delle azioni**

Con un modello, i valori degli stati sono sufficienti per determinare una policy. Se il modello non è disponibile, si preferisce stimare in base alla funzione qualità: si visitano le coppie stato-azione.

Anche in questi metodi dobbiamo mantenere l’esplorazione: gli episodi iniziano in una data coppia stato azione ed ognuna di queste ha una probabilità diversa da zero di essere selezionati.

**Monte Carlo Control**

Nel Monte Carlo control si utilizza la stessa idea del **Generalized Policy Iteration** in cui si alternano evaluation e improvement.

Per determinare la sua convergenza si assume che abbiamo osservato un numero finito di episodi e che vengano inizializzati con un inizio esplorativo.

**Monte Carlo Control – Off-policy e On-policy**

I metodi on-policy provano a valutare o migliorare la policy utilizzata per effettuare le decisioni.

I metodi off-policy provano a valutare o migliorare una policy in un istante differente a quello in cui si sono generati i dati.

**Metodi On-policy epsilon soft**

I metodi on-policy epsilon soft sono metodi di controllo che soddisfano:

Immagine che contiene testo

Descrizione generata automaticamente

Una policy greedy soft policy è:

Immagine che contiene testo

Descrizione generata automaticamente

E per preservare l’esploraizone si muova una determinata policy ad una epsilon greedy.

Nei metodi on-policy sofr greedy l’ambiente è modificato. Infatti: se nello stato s prendiamo un’azione a, allora con probabilità 1-epsilon il nuovo ambiente si comporta come il vecchio, altrimenti con probabilità epsilon si prende un’azione casuale. Il meglio che si può fare in questo nuovo ambiente con politiche deterministiche è la stessa cosa migliore che si potrebbe fare nell’

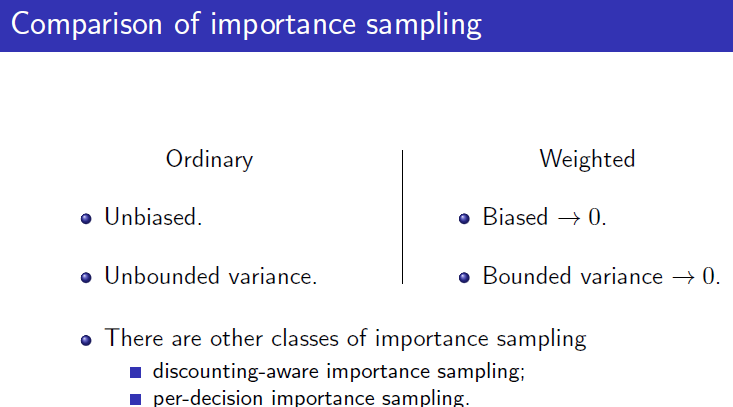
ambiente originale con le politiche "-soft. Come al solito si ha convergenza e la policy che si ottiene è ottimale.

**Metodi off-policy**

I metodi off-policy sono più generali, complessi e convergono più lentamente. Possono essere utilizzati per imparare dai dati, apprendere la policy ottima mentre si utilizza una policy esplorativa, apprendere su molteplici policy mentre se ne sta seguendo una. Infine, si riutilizza l’esperienza precedente.

In questi metodi si introduce il concetto di importance sampling che permette di valutare il valore atteso di una policy avendono un’altra: i ritorni vengono pesi secondo una certa provabilità. Il rapporto dell’importance sampling trasforma i ritorni ad avere il giusto valore atteso:





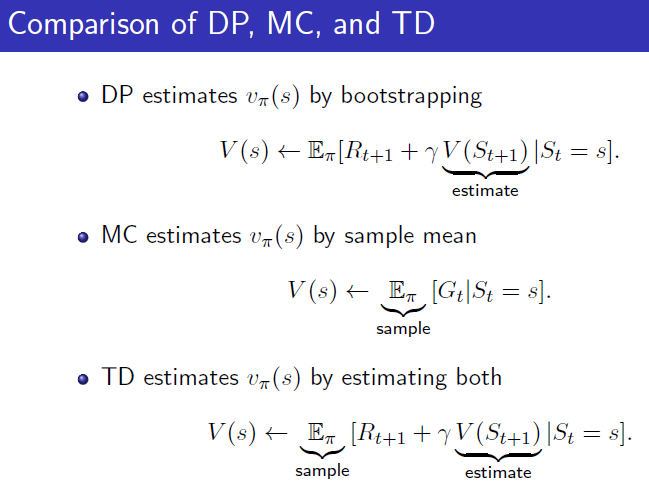
**Off – Policy Monte Carlo Control**

Il target della policy è la policy greedy rispetto alla stima della funzione valore. Si apprende dalla fine dell’episodio con azioni greedy.

**Temporal – Difference Learning**

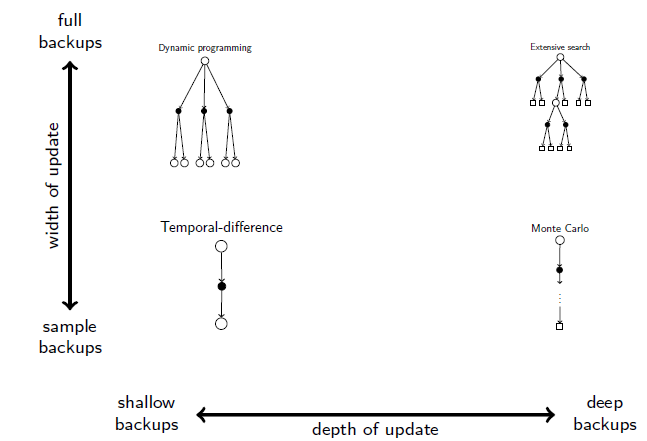
A differenza dei metodi di Monte Carlo in cui si aspetta fino a che il ritorno a seguito di una visita è noto, i metodi Temporal – Difference (TD) usanto il ritorno istantanio per apprendere:





Nel temporal difference si utilizza l’errore delta per muovere il valore della funzione valore di uno stato verso una stima migliore. L’errore non è disponibile fino all’stante t+1.

Se la funzione valore non cambia, l’errore monte carlo converge.



**Vantaggi del Temporal Difference**

* Il temporal difference può essere implementato online;
* A differenze dei metodi di monte carlo in cui si aspetta la fine dell’episodio, TD richiede un singolo passo;
* I metodi di Monte Carlo richiedono sequenze complete, TD anche con sequenze non complete;
* Monte carlo può essere applicato solo a task episodici, TD può essere utilizzato per task continui e episodici.

**Bias e Varianza degli stimatori**

* Il target montecarlo è il ritorno e non è affetto da bias;
* Il target vero di TS è R(t+1)+gamma\*v(S(t1)) ed è non affetto da bias; tuttavia il target è affetto da bias;
* Il target MC ha una varianza minore rispetto a quello di monte carlo poiché il ritorno dipende da molteplici azioni;transizioni e ricompense casuali;
* Il target TD dipende su una unica azione, transizione, ricompensa casuale.

**Vantaggi e svantaggi MC e TD**

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.

**Aggiornamento a Batch**

Supponiamo di avere una quantità finita di esperienza, possiamo utilizzare questa esperienza ripetutamente finché non si arriva a convergenza. Quindi, data una funzione valore V, si aggiorna solo una volta per ogni batch: si calcolano gli aggiornamenti in ogni istante di tempo e la funzione valore cambia una sola volta con la somma di tutti gli incrementi.

Sia MC e TD convergono deterministicamente, ma i loro risultati possono essere differenti.

**Equivalenza Certa**

Monte Carlo a batch converge alla soluzione con l’errore quadratico medio

TD a batch converge alla soluzione come se si avesse un MDP basato sui dati. Quindi si assume che la stima del processo sia nota.

**Off-policy TD prediction**

Nel TD off-policy si utilizzano i target TD generato da b per valutare la policy. Quest target vengono pesati in base all’importance sampling.

**Off-policy TD control**

Si utilizza il concetto del GPI: il TD viene utilizzato per la parte di valutazione della funzione qualità.

**SARSA**

Negli algoritmi di tipo sarsa (Stato – Azione- Reward – Stato -Azione) si utilizza TD(0) per i valori delle azioni e l’aggiornamento è effettuato in base alla quintupla (Stato – Azione- Reward – Stato -Azione). Questi algoritmi convergono con probabilità 1.

**Greedy in the limit with infinite exploration**

Una policy è detta GLIE se:

* Tutte le coppie stato azioni sono esplorate un numero infinito di volte;
* La policy converge ad una policy greedy;

**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.

**Expected SARSA**

Nell’expected SARSA si considera l’aggiornamento del Q – Learning, piuttosto che massimizzare si prende l’aspettativa della stima della funzione qualità. Questo consente di eliminare la varianza dovuta alla selezione casuale dell’azione all’istante successivo

Può essere utilizzato off-policy.

**Massimizzazione del bias**

Tutti gli algoritmi di controllo fino ad ora coinvolgono la massimizzazione:

* Q – Learning il target della policy è una policy greedy rispetto alla stima della funzione qualità;
* In SARSA la policy è spesso epsiolon-greedy.

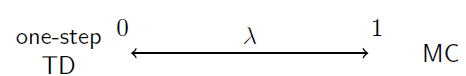
Una massimizzazione sulle stima potrebbe portare ad un bias non nullo poiché gli stessi dati sono utiliizati sia per le azioni ottimali che per i loro valori.

**Double Learning**

Il double learning è un TD control modificati in cui si dividono i dati in due insiemi e si utilizzano entrami 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.

**Tracce di Elegibilità**

Una traccia di elegibilità è un record della ricorrenza di un evento e permette di costruire un ponte tra gli eventi e le informazioni di addestramento. Quindi rappresenta un ponte fra i metodi alle differenze temporali e i metodi monte carlo:



**Metodi n-step**

Con i metodi alle differenze temproali ad un passo, lo stesso istante di tempo determina quando spesso una azione può essere cambiate e l’intervallo di tempo sul quale si effettua bootstrapping. Questo vincolo di un passo può essere rilassato.

**n-step TD prediction**

Monte Carlo effettua aggiornamenti basati sull’intera sequenza delle ricompone; TD(0) è basato solamente sulla prossima ricompensa ed effettua bootstrap: il valore del prossimo stato è usato come un proxy per le ricompense future.

**n-step target**

In questi metodi il target è il ritorno ad un passo. L’aggiornamento utilizza le ricompense e gli stati futuri; quindi occorre aspettare t+n passi per verdere la ricompensa Rt+n e calcolare il valore.

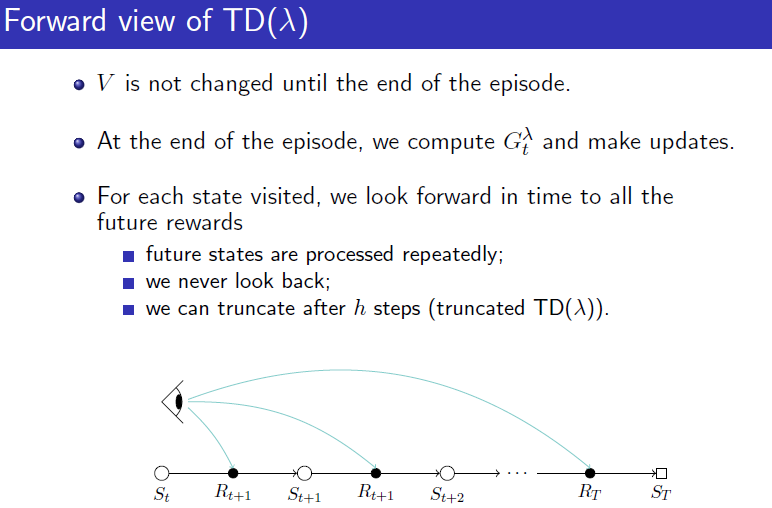
I metodi TD ad n passi convergono alle corrette predizioni.

**n-Step SARSA**

I ritorni ad n passi possono essere suddivisi in termini di valore di azioni, le azioni sono scelte seconod una politica epsilon greedy rispetto alla Q e utilizzando importance sampling si ottengono algoritmi off-policy.

Immagine che contiene testo

Descrizione generata automaticamente



La forward view non è implementabile, quindi si utilizza una visione all’indietro.

**Tracce di Elegibilità**

Le tracce di elegibilità forniscono una memoria addizionare 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.

**Algoritmo Td(λ )**

L’errore TD per la predizione stato valore è delta. Nella visione all’indietro gli aggiornamenti sono proprozionali alle tracce di elegibilità e questi incrementi possono essere effettuati sia on-line che off-line.

Immagine che contiene testo

Descrizione generata automaticamente

Se uno stato è rivisitato prima che la traccia vada a zero, con le tracce accumulative la sua elegibilita può diventare maggioere di uno. Il rimpiazzo della tracca risolve questo problema.

La traccia a duch è un compromesso tra le tracce a rimpiazzo e quelle accumulative.

**SARSA(λ)**

Nel sarsa lambda si applica la predizione TD(λ) per le coppie stato azione e l’aggiornamento della stima della funzione qualità dipende sia dall’errore TD che dalla traccia di elegibilità.

**Q(λ)**

SARSA(λ) è un metodo ci contrllo on-policy, se volessi un metodo off-policy si utilizza il Q(λ) in cui si guarda in avanti fino alla prossima azione esplorativa. Il suo aggiornamento è identico al SARSA: le tracce vengono annullate su viene prese una azione esplorativa.

Il difetto di questo algoritmo è che l’apprendimento è lento, ma leggermente più veloce del Q-learning.

**Coupling planning and learning**

I metodi indiretti di apprendimento spesso utilizzano al massimo la quantità di esperienza. I metodi diressi sono più semplici e non affetti da bias.

Quando il modello non è accurato, il planni calcorerà una policy sub-ottimale.

**Come apprendere un modello**

Immagine che contiene testo

Descrizione generata automaticamente

**Dyna-Q**

L’algoritmo Dyna-Q utilizza un modello di apprendimento deterministico con il metodo ad un passo Q-learning:

* Si campiona casualmente dalle coppie stato azione che sono state precedentemente vissute;
* Il modello ritorna come predizione l’ultimo stato e ricompensa successive osservate.

L’apprendimento e il planning differiscono solo nella sorgente della loro esperienza.

* Apprende più velocemente in ambienti deterministici;
* Se l’ambiente cambia, si può adattare;
* Il processo di planning è simile al calcolo di una policy nsub ottimale;
* L’esplorazione e lo sfruttamento entrano in conflitto poiché il primo prova a migliorare il modello mentre il secondo prende la migliore azione possibile in base al modello che si ha a disposizione.

**Dyna-Q+**

L’algoritmo Dyna - Q+ tiene traccia per ogni coppia stato azione di quanto tempo è trascorso dall’ultima visita. Più è trascorso tempo, più il modello non è corretto.

Quindi, si incoraggia il comportamento che mette alla prova le azioni non scelte da più tempo:

* La ricompensa è r;
* La transizione non è stata scelta per τ istanti di tempo;
* Il planning assume che la ricompensa è r+k√τ

**Prioritized Sweeping**

Il planning è più efficiente se simula le transizioni e gli aggiornamenti sono focalizzati su una particolare coppia stato-azione. Se le simulazioni sono generate unifiromente, allora si producono molteplici aggiornamenti non necessari.

In generale, vogliamo lavorare all’indietro da ogni stato a cui abbiamo cambiato valore:

* le coppie precedenti di quelle che sono cambiat, cambieranno più frequentemtne;
* Si assegna la priorità degli aggiornamenti in base a una misura della loro urgenza;
* Si necessita di un modello inverso.

**Trajectory Sampling**

Si distribuiscono gli aggiornamenti secondo una distribuzione on-policy:

* Si simulano individualmente le traiettorie e si effettuano gli aggiornamenti negli stati incontrati lungo la traiettoria.

Gli stati attualmente visitati sono aggiornati più spesso, mentre il resto viene ignorato.

È lo stesso concetto che avviene nella programmazione dinamica: si trova una policy ottimale sugli stati rilevanti senza visitare ogni stato indefinitamente spesso.

**Planning a tempo di decisione**

Il planning viene eseguito nello stato corrente. I valori e la policy sono specifici per il determinato stato, sono creati e cancellati una volta che sono stati utilizzati.

Questa tecnica è utile nelle applicazioni in cui non sono richieste risposte veloci.

**Ricerca Euristica**

Nella ricerca euristica si viaggia all’interno dell’albero dei possibili successori. Si utilizza un modello sub-MDP partendo dall’t-esimo istante e ci si focalizza sugli stati/azioni che seguonto immediatamente. Il nodo radice dell’albero è St.

**Algoritmi di Rollout**

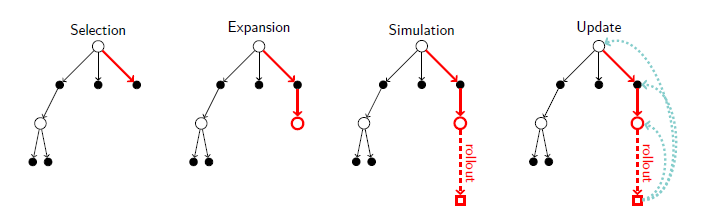
Gli algoritmi di rollout effettuano una ricerca euristica guidata da simulazioni Monte Carlo. In particolare, si simulano gli episodi da adesso con il modello. I ritorni medi delle traiettorie simulati che iniziano con ogni azioni e infine seguono una policy di rollout:

* Si simulano K episodi seguento la prima azioni a e la policy;
* Si valutano le azioni con la media del ritorno;
* Si prende l’azione che massimizza la stima della funzione qualità;

**Monte Carlo tree search**

I metodi Monte Carlo tree search usano i metodi di rollout modificando la policy:

* Si registrano i valori della stima della funzione qualità nel search tree;
* Nell’albero, prendiamo le azioni che massimizzano la stima della funzione qualità:
* Fuori dall’albero si utilizza una policy di default;
* Si applica Monte Calro all’esperienza simulata;
* Si espande la parte dell’albero che sembra promettente;
* Talvolta è possibile sostituire MC con TD.



**Predizione con Approssimazione funzionale**

Se lo spazio di stato è grande, allora i metodi tabulari non sono implementabili. Quindi, si preferisce utilizzare l’approssimazione funzionale, introducendo un vettore di pesi, si approssima il valore della funzione valore.

Cambiamenti sul vettore di pesi cambiano il valore a molteplici stati.

Questa tecnica consente di applicare il Reinforcement Learning agli MDP parzialmente osservabili.

**Targe Update**

La funzione valore V viene sostituita da v(St,wt).

**Obiettivo di Predizione**

Nei metodi tabellari, un aggiornamento in uno stato non influisce sugli altri. Questo porta ad un miglioramento per uno stato, ma un peggioramento per gli altri.

Da qui nasce l’esigenza di specificare quali stati sono maggiormente di interesse:

* Si utilizza un distribuzione μ che rappresenta il tempo speso nello stato s.

Quindi l’obiettivo è quello di massimizzare la media dell’errore quadratico medio:

Immagine che contiene testo

Descrizione generata automaticamente

**Discesa del Gradiente**

Sia w un vettore di numeri reali e v(s,w) una funzione differenziabile a valori reali. Inoltre, assumiamo che gli stati sono presentati con un distribuzione μ e che sia possibile misurare il valore vero della funzione valore.

L’aggiornamento di w, usando il gradiente discendente stocastico è:

Immagine che contiene testo

Descrizione generata automaticamente

**Approssimazione del gradiente discendente stocastico**

Sia U una approssimazione stocastica della funzione valore.

Immagine che contiene testo

Descrizione generata automaticamente

**Bootstrapping con l’approssimazione funzionale**

MC SGD converge poiché U è una stima senza bias della funzione valore. Applicando il bootstrapping il target dipende dai pesi, acquista bias e non è un vero SGD.

**Metodi Lineari**

L’approssimazione funzionale è lineare rispetto ai pesi.

Nei metodi lineari si associa per ogni stato un vettori di feature x(s). La stima diventa quindi:

Immagine che contiene testo

Descrizione generata automaticamente

**Selezione delle Feature**

Al fine di aggiungere una conoscenza maggiore del dominio è utile scegliere appropriatamente le feature per i task. Queste dovrebbero corrispondere agli aspetti dello spazio di stato lungo il quale è possibile generalizzare.

Le forme lineari non possono render conto di qualsiasi interazioni tra le features se codificati separatamente.

**Features**

* Polinomiali: crescono esponenzialmente con la dimensione dello spazio di stato;
* Fourier:
  + sono utili per funzioni periodiche o domini limitati.
  + Se approssimati tra o e pi/2 si può utilizzare il coseno;
  + Crescono esponenzialmente con la dimensione dello spazio di stato;
* Coarse coding:
  + Si divide lo spazio delle feature in cerchi. Ognuno di questi ha il proprio peso;
  + i pesi di tutti i cerchi che si intersecano in uno stato sono influenzati;
  + grandi feature danno una maggiore generalizzazione;
  + accuratezza viene controllata dal numero totale di features.

**Tile Coding**

* Si utilizzano delle griglie per partizionare lo spazio di stato;
* Si sovrappongono le griglie, senza far sovrapporre i tile;
* In ogni tile si trova una feature;
* Si ha una maggiore generalizzazione se l’offset è asimmetrico;
* Si preferisce usare differenti tile di divese forme in ogni griglia;
* Le funzioni di hash permettono di ridurre il problema del curse of dimensionality;

**Radial Basis Function**

* Sono una generalizzazione del coarse coding;
* Una feature è continua nell’intervallo 0,1
* Si utilizzano funzioni gaussiane
* Il costo computazionale è alto

**Step-size**

Teoricamente la dimensione del passo dovrebbe decrementare nel tempo, ma una buona regola è quello di immpsotare il parametro di ste come:

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

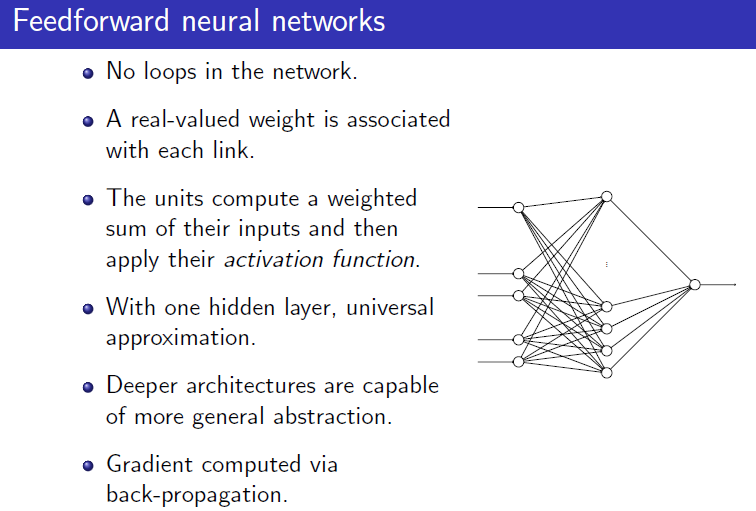


Immagine che contiene testo

Descrizione generata automaticamente

**Adattamento delle tracce di eleggibilità all’approssimazione funzionale**

Le tracce di eleggibilità possono essere utilizzate con l’approssimazione funzionale. Nell’algoritmo off-line a λ

-ritorni, alla fine dell’episodio, si effettuano una sequenza di aggiornamenti:



Le tracce di elegibilità sono vettori z che costituiscono la memoria a breve termine ed influenzano il peso dei vettori che quest’ultimi determinano la stima della funzione valore.

**Immagine che contiene testo

Descrizione generata automaticamente**

**Controllo con approssimazione funzionale**

Vogliamo costruire una approssimazione della funzione qualità.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente