REINFORCEMENT LEARNING

ANTONIO TURCO

4 marzo 2021

INDICE

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

ABSTRACT

Il reinforcement learning (spesso abbreviato con **RL**) è una disciplina dell'IA nata negli anni '50 che sta riguadagnando un discreto successo dopo una serie di articoli pubblicati da DeepMind riguardo l'uso di *reti neurali* all'interno dei processi decisionali che deve effettuare l'agente RL. In questa relazione si vogliono comprendere le basi su cui si fonda questa disciplina partendo dagli algoritmi classici come **REINFORCE** per arrivare alle tecniche più moderne di **Deep Q-Learning**. Inoltre verranno provati un insieme di ambienti giocattolo forniti dalla libreria *gym.ai* sulle diverse tecniche per verificarne sperimentalmente pregi e difetti.

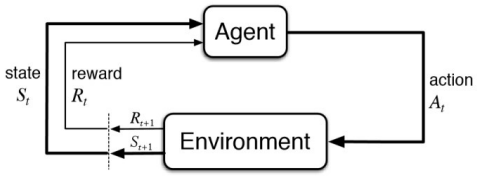


Figura 1: Schema che riassume l'approccio applicato dal reinforcement learning. Si può notare come l'approccio ciclico comporti che l'agente possa influenzare le osservazioni che riceverà dall'ambiente. (Immagine presa da https://www.kdnuggets.com).

1 INTRODUZIONE

Il reinforcement learning è una disciplina dell'IA che ha l'obiettivo di definire il comportamento di un agente attraverso un apprendimento **trial and error**. Ovvero si cerca di ottenere l'ottimizzazione di una funzione obiettivo commettendo anche degli errori inizialmente, per cercare di capire come funziona il dominio in cui l'agente opera. Tale approccio è differente dalle metodologie classiche di apprendimento (supervisionato e non) perché non ha alcun dataset su cui lavorare normalmente e la scelta ottima spesso non è nota a priori. Infatti il problema può essere definito da tre componenti principali:

1. **Agente**, entità astratta che può compiere azioni in un ambiente e che cerca di massimizzare il guadagno
2. **Ambiente**, che caratterizza lo spazio in cui si muove l'agente. In particolare fornisce di solito uno stato con cui l'agente può prendere decisioni
3. **Ricompense**, forniscono un segnale di guadagno o perdita all'agente rispetto alle azioni e le osservazioni precedenti

Le seguenti intuizioni possono essere viste in maniera più intuitiva nello schema della figura 1, dove si può osservare come l'agente e l'ambiente si influenzino a vicenda per via della definizione **ciclica** del problema. Inoltre si può facilmente osservare che l'unica fonte di feedback per la stima della funzione obiettivo è il segnale di ricompensa fornito dall'ambiente (anche detto *reward*), che normalmente è un valore scalare che può essere sia negativo che positivo. Tale valore non è sotto il controllo diretto dell'agente e viene fornito dall'ambiente, quindi spesso può capitare che il guadagno ricevuto si riferisca in realtà ad azioni passate.
Infine è utile osservare che l'agente non può interagire in altro modo con l'ambiente se non con le azioni prestabilite, infatti non può interrogare il modello dell'ambiente per ottenere informazioni aggiuntive. Quindi l'ambiente per queste applicazioni deve essere visto come una **scatola nera**.

Per finire questa introduzione si vogliono evidenziare le problematiche principali che vengono affrontate durante la costruzione di un agente attraverso l'uso di tecniche come quelle data dall'apprendimento per rinforzo.

1.1 Ricerca politica ottima

Date le precedenti definizioni del problema che si vuole risolvere, è naturale concentrarsi sulla ricerca della politica (o *policy*) migliore che può seguire un'agente. Ovvero si vuole trovare "l'algoritmo" che prende sempre la decisione corretta. Tale ricerca è il nucleo principale di tale dominio, infatti tutti gli algoritmi che verranno presentati successivamente cercano in maniere differenti di individuare la politica migliore per l'agente nell'ambiente in cui si trova. Tutte le politiche che verranno utilizzate saranno politiche **stocastiche**. In questa maniera si cercherà semplicemente di trovare la distribuzione di probabilità ottima delle azioni che si possono com-

piere nell'ambiente. Si noti che tale compito degenererebbe in un apprendimento supervisionato se tali distribuzioni fossero note a priori.

1.2 Da grandi poteri derivano grandi responsabilità

Una delle difficoltà maggiori nella ricerca della politica ottima risiede nella difficoltà di assegnare un certo reward all'azione che ha effettivamente generato tale ricompensa. Infatti se si prendesse una partita di scacchi per esempio, se la ricompensa arriva solo al termine della partita, allora è difficile capire quanto ogni mossa sia stata decisiva per la vittoria finale.

1.3 Esplorare o ottenere un vantaggio?

Una delle domande principali che l'agente si deve porre durante l'apprendimento è: meglio provare a sfruttare la conoscenza che ho dell'ambiente per massimizzare il mio guadagno o è meglio provare ad esplorare l'ambiente e quindi provare ad apprendere più informazione possibile? Ovviamente massimizzare solo uno dei due comportamenti significa avere una policy inefficace. Nel primo caso infatti non si avrebbe abbastanza conoscenza del dominio e quindi non sarebbe possibile effettuare nessuna decisione ottima, mentre nel secondo caso la sola esplorazione non potrebbe, per definizione, portare alla politica ottima cercata. Quindi questo aspetto introduce un **trade-off** che dovrà essere bilanciato saggiamente per ottenere le prestazioni ottimali per il nostro agente.

1.4 Architettura delle reti neurali

Se si vuole sfruttare, in tale contesto, una rete neurale bisogna porre attenzione principalmente sui layer finali dell'architettura. Infatti tali layer devono essere gestiti in modo tale che restituiscano un output probabilistico rispetto alle azioni disponibili o una misura dell'azione migliore in un certo contesto. L'utilizzo delle probabilità rispetto a dei valori deterministici permette all'agente di esplorare diverse possibilità nonostante venga imposta un azione ottima con un alta percentuale.

2 POLICY GRADIENTS

Il primo approccio che si è provato ad utilizzare per implementare un sistema di reinforcement learning riguarda l'uso dei *policy gradients*. Tale tecnica cerca di usare le idee classiche dell'ottimizzazione viste già nei sistemi supervisionati ma in un contesto dove la funzione target non è nota a priori.

2.1 Assegnazione reward

Prima di affrontare l'algoritmo di ottimizzazione, è necessario capire come tale tecnica risolve il problema dell'assegnazione della ricompensa. Per fare ciò si definisce un *fattore di sconto* γ che indica all'agente quanto è importante il futuro per l'azione corrente. A questo punto l'azione viene valutata con una sommatoria dei reward futuri che riceverà applicando ad ogni step il fattore γ. In maniera più formale si può scrivere che il guadagno atteso G_t per un azione a dopo l'istante t è:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

Dove i valori R_t rappresentano i reward ottenuti all'istante t. Da questa equazione si può facilmente intuire che con $\gamma \to 0$ l'agente ignorerà ciò che avviene nel futuro

per l'azione corrente, mentre per $\gamma \to 1$ il futuro avrà la stessa importanza del presente per l'azione corrente secondo l'agente.

2.2 Algoritmo REINFORCE

L'idea è quindi di ottimizzare i parametri della politica che un agente può sfruttare seguendo l'informazione ottenuta dai gradienti. Dati gli andamenti dei gradienti si cerca di aumentare la probabilità di un azione se tale azione ottiene un reward positivo mentre di decrementarla in caso negativo. Si riporta qui sotto l'intuizione dell'algoritmo *REINFORCE* che applica una sorta di gradient ascent ai parametri del modello di rete neurale in input:

1. Si lascia giocare la rete neurale per una serie di episodi e si memorizzano i gradienti che renderebbero le azioni scelte ancora più probabili. Tali gradienti in questa fase sono solo memorizzati ma non ancora applicati
2. Si calcola il guadagno che ogni azione ha portato usando l'equazione 1
3. Si moltiplicano i gradienti con il vantaggio corrispondente calcolato prima. In questa maniera se un azione ha dato una guadagno positivo, allora il sistema cercherà di sfruttarla maggiormente.
4. A questo punto si possono applicare i gradienti calcolati precedentemente

2.3 Risultati

L'algoritmo è stato testato con due ambienti giocattolo presenti nella libreria *gym*, chiamati "CartPole-v1" e "MountainCar-v0".

CART POLE Il gioco consiste nel mantenere in equilibrio un bastone posto sopra un carrello. L'agente ha il controllo del carrello e l'ambiente è definito da uno spazio osservabile composto da 4 parametri: posizione carrello, velocità carrello, angolo bastone, velocità angolare bastone. Le azioni possibili sono due: muovere il carrello a sinistra o muovere il carrello a destra. Dopo ogni azione viene sempre dato reward $+1$ finché il bastone non cade dal carrello. La simulazione va avanti per 200 frame e l'agente ottimo dovrebbe avvicinarsi il più possibile ad ottenere un reward pari a 200 alla fine dell'episodio.
In questo caso l'apprendimento basato su policy gradient si è dimostrato efficace come si vede dalla figura 2. Infatti con un numero non troppo elevato di iterazioni da 15 episodi l'una, l'agente è riuscito ad ottenere una policy che raggiunge prestazioni ottimali. Inoltre come si vede in tabella 1 questo metodo è riuscito a fare molto meglio di una poltica random o di una politica fissa data a priori.
L'aspetto più interessante riguarda la forte instabilità tra un'iterazione e la successiva, dovute principalmente ai vari tentativi di esplorazione effettuati dall'agente durante le varie iterazioni.

MOUNTAIN CAR Il gioco consiste nel raggiungere la vetta della collina cercando di far "ondeggiare" una macchina a destra e sinistra. L'agente ha quindi il controllo di una macchina con tre possibili azioni: non fare niente, spostarsi a sinistra e spostarsi a destra. Lo spazio osservabile è caratterizzato dalla posizione della macchina e dalla velocità attuale della macchina. Il valore di reward restituito è sempre -1 se non quando la macchina raggiunge la cima della collina, in tal caso il reward sarà di 0.5 e l'episodio verrà terminato con successo. Su questo environment tale approccio si è rivelato fallimentare, principalmente per l'aumento del numero di parametri e dai reward che non danno molte informazioni. L'aumento del numero di parametri ha reso più costoso l'addestramento della rete neurale mentre il fatto che il reward positivo venga dato solo quando si vince, rende impossibile migliorare negli episodi negativi, visto che viene restituita sempre la stessa loss. In questo caso

	Random policy	Hard coded policy	Policy gradient
mean reward	41.17	48.41	197.52

Tabella 1: Guadagno medio nella palestra *CartPole-v1*

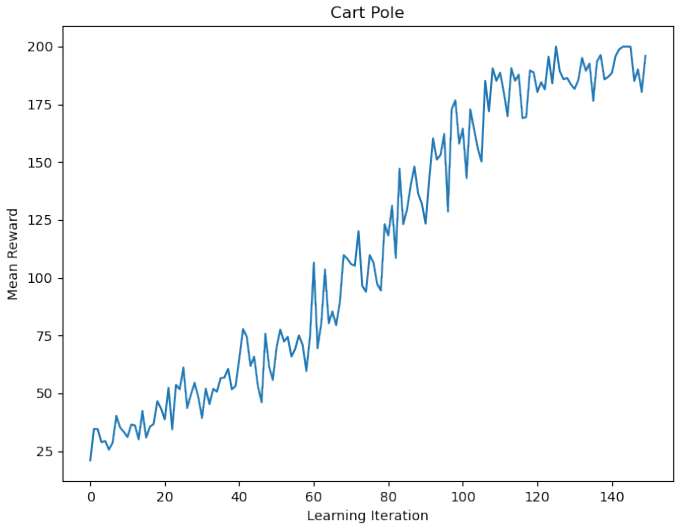


Figura 2: Grafico che mostra l'andamento dell'apprendimento della policy ottima per il problema CartPole

non si riportano grafici visto che il guadagno ottenuto è sempre rimasto fisso a -200.

2.4 Problematiche

Osservando i risultati precedenti quindi è utile riassumere le problematiche principali dell'algoritmo di policy gradient.

1. **Bassa scalabilità**, basta introdurre qualche parametro in più per rendere proibitivo il calcolo del training loop;
2. **Necessita di reward postivi**, se non riesce ad ottenere con policy random valori di reward positivi sin da subito, non riesce a capire in quale direzione bisogna muoversi per migliorare i pesi della rete neurale in poco tempo.

Tali difficoltà tecniche possono essere superate attraverso l'uso di altre tecniche come il Q-Learning che verrà affrontato successivamente.

3 PROCESSI DECISIONALI DI MARKOV

Prima di entrare nel dettaglio del Q-Learning è necessario introdurre il formalismo dei *processi decisionali di Markov* (spesso abbreviati con *MDP*). Un processo decisionale di Markov permette la descrizione formale dell'ambiente in cui un agente si può muovere, solo nel caso di ambienti completamente osservabili.
Normalmente è rappresentato come un grafo, i cui nodi sono gli stati e gli archi sono pesati con la probabilità di andare in un determinato stato, come nell'esempio della figura 3.
L'idea principale che distingue questi grafi da quelli classici, è quella di evitare di memorizzare tutta la storia di esecuzione di un episodio e salvare solo lo stretto

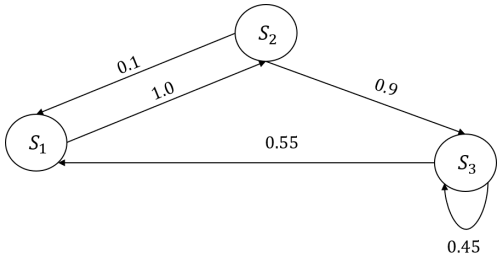


Figura 3: Esempio di Markov Decision Process dove ogni transizione viene eseguita con la probabilità riportata sull'arco

necessario per effettuare una decisione ottima. Questo viene formalizzato con l'uso degli stati *markoviani* che vengono indicati con S_t e sono formalizzati come segue:

Definizione 1. Uno stato S_t è detto markoviano se e solo se:

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, \ldots, S_t]$$

Con questa formalizzazione si va quindi ad evidenziare che uno stato, che gode della proprietà di essere markoviano, riassuma tutta l'informazione rilevante per l'agente per effettuare una decisione ottima. Infatti la probabilità P di transizione tra uno stato e l'altro resta uguale nonostante sia stata *gettata* la storia dell'intero episodio eccetto lo stato corrente.
Prima di poter utilizzare tale strumento matematico per la risoluzione di problemi del dominio RL, bisogna introdurre il concetto di azioni all'interno di questi MDP.

Definizione 2. Un processo decisionale di Markov è una tupla $\langle S, A, P, R, \gamma \rangle$:

1. S è un insieme *finito* di stati
2. A è un insieme *finito* di stati
3. P è una matrice di transizione di stato che ha il seguente comportamento:

$$P^a_{ss'} = P[S_{t+1} = s' \mid S_t = s, A_t = a]$$

4. R è la funzione di guadagno espressa come:

$$R^a_s = E[R_{t+1} \mid S_t = s, A_t = a]$$

5. $\gamma \in [0, 1]$ è il fattore di sconto

Attraverso questa definizione è possibile usare le equazioni di Bellman che vengono riportate nell'equazione 2 per risolvere il processo decisionale.

$$v^*(s) = \max_a R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} v^*(s') \tag{2}$$

Tale equazione può essere utilizzata per trovare i valori ottimi per ogni stato e quindi individuare successivamente la policy π^* ottima semplicemente scegliendo per ogni stato l'azione che massimizza il value score v, in maniera *greedy*.
Tale approccio funziona in maniera ottima se tutti i dati sono *noti a priori*, assunzione che **non** è quasi mai vera nei problemi reali di reinforcement learning. Infatti non si possono avere a priori i valori della transazione di probabilità $P^a_{ss'}$ e della funzione di reward R^a_s. Tali grandezze devono essere stimate durante l'apprendimento attraverso l'uso di algoritmi come il Q-learning.

4 Q-LEARNING

Q-Learning è un algoritmo iterativo di apprendimento dove le probabilità di transizione e i reward sono inizialmente **ignoti**. Q-Learning parte osservando un agente

Figura 4: Screenshot del gioco *Breakout*

giocare nell'ambiente (ad esempio in maniera random) per migliorare gradualmente le sue stime dei Q-value $q(s, a)$. Una volta che ha definito i Q-value ottimi, la politica ottima π^* viene individuata scegliendo l'azione che ottiene il Q-value massimo. La stima della funzione $q(s, a)$ avviene sfruttando l'equazione 3:

$$q(s,\ a) = (1 - \alpha) \cdot q(s, a) + \alpha\,(r + \gamma \cdot \max_{a'}\ q(s', a')) \tag{3}$$

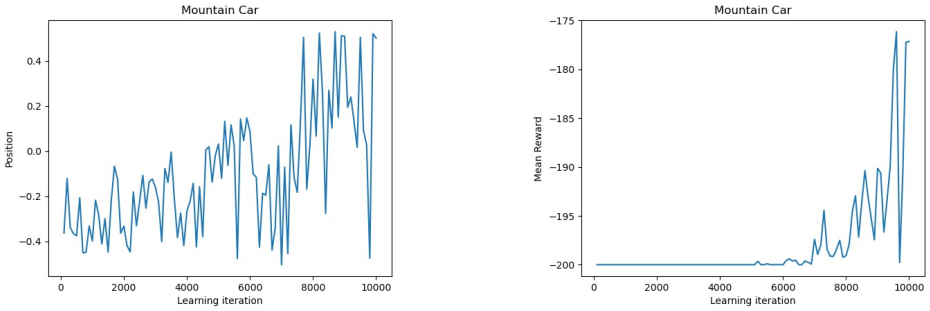
Ovvero per ogni coppia $(s,\ a)$, l'algoritmo tiene traccia della media corrente dei reward r che un agente può ottenere una volta che lascia lo stato s eseguendo l'azione a più la somma dei guadagni futuri scontati assumendo che si comporti ottimamente.

4.1 Deep Q-Learning

L'approccio Q-Learning quando applicato alle reti neurali profonde, prende il nome di *Deep Q-Learning* (spesso abbreviato con DQN). Tale rete deve prendere in input lo stato osservato dall'ambiente e in output deve avere tanti nodi quante sono le azioni disponibili per l'agente. I nodi d'output, dato un stato s input, conterranno il valori approssimati di $q(s,\ a)$ dove il valore per l'azione a_i verrà mappato con il nodo d'output i-esimo.
Una particolarità di questa tipologia di apprendimento riguardo l'utilizzo di una ϵ*-policy*. Ovvero invece che scegliere sempre l'azione ottima secondo la Q-table, si sceglie con probabilità ϵ un'azione random. Tale parametro di solito viene posto a 1 per poi farlo decrescere a 0, mano mano che la rete apprende la funzione Q-value ottimale.
Un accorgimento utile riguarda la possibilità di **memorizzare le esperienze** ottenute dall'agente sull'ambiente in un *replay buffer*, così da poter sfruttare anche conoscenza passata che può essere utile in certe situazioni. Le esperienze poi saranno campionate a random in modo tale che la correlazione tra queste sia minore e il passo di aggiornamento dei pesi può ottenere più informazione possibile da questa varietà.

4.2 Risultati

MOUNTAIN CAR Attraverso l'uso del DQN è stato possibile risolvere tale gym. Come si può vedere dal grafico 5a, l'algoritmo riesce a raggiungere una soluzione che lo porta sempre al goal (valore *position* $= 0.4$). L'aspetto più critico di questa metodologia sono i grandi tempi richiesti per l'addestramento della rete neurale come mostrato nella figura 5b. Infatti per questa palestra le prime 6000 iterazioni circa sono servite all'agente per esplorare l'ambiente in maniera random. Le successive iterazioni invece sono state necessarie per exploitare l'ambiente attraverso l'ottimizzazione dei pesi della rete. Si nota una certa instabilità anche in questo caso, dovuto al fatto che la rete in alcune fasi dell'apprendimento sembra dimenticare



(a) Convergenza posizione ottima MountainCar **(b)** Andamento reward MountainCar

Figura 5: Grafico che mostra andamento di DQN su gym MountainCar-vo

come comportarsi in certi stati. Per evitare questo si potrebbe aumentare il numero di campioni che viene memorizzato nel replay buffer o provare ad aumentare il numero di iterazioni della fase di apprendimento. Inoltre sarebbe utile memorizzare i pesi della rete con i risultati migliori, così che alla fine del training loop si otterrà la rete che riesce ad ottimizzare il processo decisionale sull'ambiente in input.

BREAKOUT In questa gym l'agente deve imparare a giocare al famoso gioco *Atari* dove bisogna abbattere un muro posto sulla parte superiore dello schermo come mostrato nella figura 4. L'agente può controllare una barra orizzontale che permette di far rimbalzare la pallina verso il muro di mattoni. La risoluzione della gym oltre a sfruttare una DQN deve anche introdurre una rete convoluzionale per estrarre informazione dalle immagini che vede frame per frame. Sfortunatamente la risoluzione di questa gym non è stata raggiunta dall'agente DQN. La motivazione principale è probabilmente legata al fatto che richiede delle risorse di calcolo particolarmente elevate. Infatti l'apprendimento ha richiesto più di qualche ora e nonostante tutto il modello ottenuto non è riuscito ad andare oltre un punteggio massimo di 3. L'intuizione che la problematica sia legata ad una mancanza di risorse è supportata dal fatto che anche negli esperimenti di DeepMind hanno avuto il bisogno di processare più di 1 milione di frame prima di raggiungere risultati soddisfacenti, calcolo che sia su Colab che su un pc personale risulta proibitivo.

5 CONCLUSIONI

Da questa esperienza si può comprendere come il reinforcement learning possa essere una metodologia utile per risolvere problemi di **controllo di un agente**. Infatti basta definire l'ambiente, le ricompense e le azioni e attraverso metodi di ottimizzazione, l'agente, può comprendere quale è la strategia ottima per risolvere il problema. L'approccio che offre è di tipo **generale** e può essere applicato in svariati contesti (es. giochi, sistemi di raccomandazione, investimenti finanziari, etc...).
La fase di apprendimento per problemi di dimensioni non piccole richiede **molto tempo per convergere** e forti risorse computazionali per raggiungere una strategia ottima. Tale difetto rende difficile fare tuning degli iper-parametri perché ogni addestramento può richiedere anche giornate intere.
Tale problematica può essere però tamponata **introducendo della conoscenza** all'interno dell'agente per poterlo inizializzare in maniera più intelligente. Inoltre si possono modificare i reward all'interno del training loop per fargli seguire una strada che già si conosce per essere quella ottimale e a quel punto lasciare che l'agente completi la strategia per la parte che è ancora ignota.

RIFERIMENTI

[1] David Silver. Lectures on reinforcement learning. URL: https://www.davidsilver.uk/teaching/, 2015.

[2] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009.

[3] Aurlien Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2nd edition, 2019.