Ingegneria della conoscenza

Frozen-Lake

Indice

1. Introduzione

In Frozen-Lake, l'agente si trova in un ambiente 4x4 o 8x8 L'ambiente rappresenta un lago ghiacciato con alcuni buchi da evitare.

L'obiettivo è raggiungere il goal, un frisbee da recuperare. L'agente sceglierà un azione da compiere ma non si muoverà sempre nella direzione scelta perché il ghiaccio è scivoloso. L'ambiente è rappresentato come segue:

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

Tabella 1: Esempio di griglia 4x4

S: Punto iniziale, sicuro
F: Superficie ghiacciata, sicuro
H: Buco, non sicuro
G: Obiettivo, posizione del frisbee

Un episodio termina quando l'agente raggiunge l'obiettivo oppure cade in un buco. Riceve una ricompensa pari a 1 se raggiunge l'obiettivo e zero altrimenti.

2. Reinforcement Learning (Aprendimento per rinforzo)

Il Reinforcement Learning (RL) è una area del machine learning che si concentra su come un agente può agire in un ambiente per massimizzare le ricompense che riceve. Un agente di RL deve determinare quali azioni svolgere in base alle sue percezioni e alle ricompense che riceve.

Utilizzando un gioco come esempio di ambiente, il RL si concentra su come il giocatore può scegliere di agire (per esempio muoversi in una certa direzione) per massimizzare una ricompensa che in questo caso potrebbe essere il punteggio.

2.1 Markov Decision Processes (Processo decissionale di Markov)

I processi decisionali di Markov (MDP) formalizzano il processo decisionale sequenziale che è alla base del RL.

In un MDP, l'agente interagisce con l'ambiente in cui si trova. Le interazioni sono sequenziali nel tempo. Ad ogni istante T , l'agente ottiene una rappresentazione dello stato dell'ambiente. Data questa rappresentazione, l'agente decide l'azione da compiere. L'ambiente quindi entra in un nuovo stato e l'agente riceve una ricompensa che è una conseguenza della sua azione precedente.

In un MDP abbiamo:

- Un agente
- Un ambiente
- Gli stati
- Le azioni possibili
- Le ricompense

Il processo di selezionare una azione da un dato stato, transitare verso un nuovo stato e ricevere una ricompensa, avviene ciclicamente, creando una sequenza di stati, azioni e ricompense. L'obiettivo dell'agente è massimizzare la somma delle ricompense che riceve scegliendo delle azioni in degli stati precisi. Ciò significa che l'agente dovrebbe massimizzare non solo la ricompensa immediata, ma le ricompense accumulate nel tempo.

In un MDP abbiamo:

- S: un set di stati
- A: un set di azioni
- R: un set di ricompense

Ad ogni istante $t = 0, 1, 2, 3...$ l'agente si trova in uno stato $S_t \in S$ e seleziona una azione $A_t \in A$ che formano una coppia stato-azione (S_t, A_t). Possiamo pensare alla ricezione di una ricompensa come una arbitraria funzione f che mappa coppie stato-azione su ricompense. In un dato istante t abbiamo che:

$$f(S_t, A_t) = R_{t+1}$$

Si formerà dunque una sequenza del tipo:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4...$$

Gli insiemi S e R sono finiti, le variabili aleatorie R_t e S_t hanno delle distribuzioni di probabilità definite. Tutti i possibili valori che possono essere assegnati a R_t e S_t hanno delle probabilità associate che dipendono daS_{t-1} e da A_{t-1}, stato precedente e azione effettuata al tempo $t - 1$.

Per esempio, supponiamo che $s' \in S$ e $r \in R$. C'è qualche probabilità che $S_t = s'$ e $R_t = r$. Questa probabilità è determinata dallo stato precedente $s \in S$ e l'azione $a \in A(s)$, dove $A(s)$ è il set di azioni che si possono effettuare nello stato s.

3. Expected Return

L'obiettivo di un agente in un Processo decissionale di Markov è quello di massimizzare le sue ricompense cumulative. Abbiamo bisogno di un modo per aggregare e formalizzare queste ricompense.

Per questo, si introduce il concetto di expected return. Si può pensare al return come la somma delle future ricompense. Quindi possiamo definire il ritorno G al tempo t come:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} ... + R_T$$

dove T è il passo di tempo finale.

L'obiettivo dell'agente è cioè quello di massimizzare l'expected return delle ricompense. L'expected return è ciò che guida l'agente nel prendere le decisioni.

Questa definizione va bene per task "episodici", cioè in quei task che hanno un tempo finito T. Esistono anche dei task continui che non hanno limite di tempo.

4. Policies e Value Functions

4.1 Policies

Una policy o politica è una funzione che ha in input uno stato e restituisce la probabilità di selezionare ogni azione possibile da quello stato. Indichiamo una policy con π. Se un agente segue una policy π allora nell'istante t, $\pi(a \mid s)$ è la probabilità di scegliere l'azione a se ci si trova nellostato s. La policy risponde alla domanda "Quanto è probabile per un agente selezionare una data azione da un dato stato?".

4.2 Value Functions

Una Value Function è una funzione degli stati o di coppie stato-azione che stima quanto è buono per un agente essere in un dato stato o quanto è buono scegliere una specifica azione partendo da uno stato specifico. Le Value Functions sono definite rispetto alla policy che un agente segue.

4.2.1 State-Value Function

La funzione State-Value per la policy π, denotata come v_π , ci dice quanto è buono essere in uno stato s per un agente che segue la policy π. Il valore di uno stato s per la policy π è l'expected return che si ha partendo dallo stato s al tempo t e poi seguendo la policy π.

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

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

dove γ è il discount rate, compreso nell'intervallo [0, 1] che serve a considerare maggiormente i reward a breve termine rispetto a quelli a lungo termine.

4.2.2 Action-Value Function

La funzione Action-Value per la policy π, denotata come q_π , ci dice quanto è buono per un agente scegliere una specifica azione a parendo da un dato stato s seguendo la politica π.

$$q_\pi = E_\pi[G_t \mid S_t = s, A_t = a]$$

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

La Action-Value Function q_π è chiamata Q-Function e il suo output per una coppia stato-azione è chiamato Q-value, dove la lettera Q sta per qualità di una coppia stato-azione.

4.3 Policy Ottimale

L'obiettivo degli algoritmi di RL è trovare la policy che porterà l'agente a guadagnare molte ricompense se l'agente la segue. Una policy ottimale porta l'agente ad ottenere più ricompense di 0qualunque altra policy. Una policy π è considerata migliore o uguale alla policy π se l'expected 0return di π è maggiore o uguale a quello di π per tutti gli stati.

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s) \forall s \in S$$

4.4 Funzione State-Value Ottimale

La funzione State-Value ottimale, denotata con q_* è così definita:

$$v_*(s) = \max_\pi \ v_\pi(s) \forall s \in S$$

v_* in output ha l'expected return più alto che si può ottenere seguendo una policy per ogni stato.

4.5 Funzione Action-Value Ottimale

La funziona Action-Value ottimale, denotata con q_* ed è così definita:

$$q_*(s, a) = \max_\pi \ q_\pi(s, a) \forall s \in S \land a \in A(s)$$

q_* in output ha l'expected return più alto che si può ottenere seguendo la policy che lo massimizza per ogni coppia stato-azione.

4.6 Equazione di ottimalità di Bellman per q_*

Una proprietà fondamentale di q_* è quella che deve soddisfare questa equazione:

$$q_*(s,a) = E[R_{t+1} + \gamma \max_{a'} q_*(s',a')]$$

Questa equazione è chiamata Equazione di ottimalità di Bellman. Data una coppia stato-azione (s, a), al tempo t, l'expected return partendo dallo stato s, selezionando l'azione a, e poi seguendo la policy ottimale (cioè il Q-value della coppia) è la ricompensa che si ottiene selezionando l'azione a nello stato s, cioè Rt+1, più il massimo expected return che può essere raggiunto da una qualsiasi tra le prossime possibili coppie stato azione (s0 , a0) scontato di γ. Siccome l'agente sta seguendo una policy ottimale, lo stato successivo s0 sarà lo stato da cui la migliore prossima azione a0 può essere scelta al tempo t + 1, dove per migliore azione si intende quella che massimizzerà l'expected return.

5 Q-Learning

Il Q-Learning è una tecnica in grado di risolvere il problema di trovare una policy ottimale per un MDP. L'obiettivo del Q-Learning è trovare una policy ottimale in modo che il valore atteso della somma delle ricompense di tutti gli step successivi sia il massimo possibile. Il Q-Learning cerca una policy ottimale imparando i Q-values ottimali per ogni coppia stato-azione.

5.1 Value-iteration

L'algoritmo Q-Learning aggiorna iterativamente i Q-values per ogni coppia stato-azione utilizzando l'Equazione di Bellman fino a quanto la Q-function converge verso la Q-function ottimale, q_*.

5.2 Un esempio: Frozen-Lake

In Frozen-Lake, l'agente vuole recuperare il frisbee senza cadere nei buchi. L'agente può muoversi a destra, sinistra, in alto o in basso nell'ambiente. Queste sono le azioni. Lo stato è determinato dalla cella della griglia in cui si trova l'agente. L'agente riceve una ricompensa (+1) se raggiunge il goal, mentre in tutti gli altri casi non riceve alcuna ricompensa. Se cade nel buco o raggiunge il goal, l'episodio termina. All'inizio dell'episodio l'agente non ha idea di quali azioni sia meglio scegliere a partire da un dato stato. Conosce solo lo stato corrente e le azioni possibili e non sa se una data azione scelta a partire da un dato stato gli conferirà una ricompensa positiva o negativa. I Q-values vengono inizializzati a 0 per ogni coppia stato-azione in quanto l'agente non conosce nulla riguardo l'ambiente in cui si trova. I Q-values verranno aggiornati iterativamente utilizzando la Value iteration.

5.3 Q-table

La Q-table è una tabella $|\,azioni\,| \times |\,stati\,|$ in cui l'agente memorizza i Q-values per ogni coppia stato-azione. Quando la Q-table viene aggiornata l'agente può utilizzarla per scegliere l'azione con il Q-value più alto a partire dallo stato in cui si trova.

5.4 Episodi

L’agente per imparare dovrà giocare al gioco più volte. Ogni partita che l’agente gioca si chiama episodio. L’agente potrebbe scegliere le azioni basandosi sui valori memorizzati nella Q-table, ma durante il primo episodio tutti i Q-values sono settati a 0. L’agente dovrebbe avere un modo per esplorare l’ambiente prima di poter sfruttare la Q-table. Per questo vengono introdotti i concetti di Exploration e Exploitation.

5.5 Exploration vs. Exploitation

In ogni episodio, in base allo stato in cui si trova, l’agente seleziona l’azione con il Q-value più alto stimato nella Q-table. Dal momento che, durante il primo episodio, nello stato iniziale tutti i Q-values sono inizializzati a zero, non c’è nessun modo per l’agente di differenziarli e quindi non sa quale azione eseguire per prima. Inoltre, negli stati successivi, selezionare l’azione con il più alto Q-value per lo stato corrente non è sempre la scelta migliore. Perciò abbiamo bisogno di un trade-off tra exploration ed exploitation per scegliere le nostre azioni. Questo trade-off è ottenuto utilizzando la epsilon greedy strategy.

5.6 Epsilon greedy strategy

Per avere il bilanciamento giusto tra exploration ed exploitation, usiamo la cosiddetta epsilon greedy strategy. Con questa strategia, definiamo un exploration rate e che inizialmente impostiamo ad 1. Questo exploration rate è la probabilità che il nostro agente esplorerà l’ambiente piuttosto che sfruttare la Q-table scegliendo sempre l’azione migliore. Con e = 1 , è sicuro al 100% che l’agente inizierà esplorando dell’ambiente.
All’inizio di ciascun nuovo episodio, l’exploration rate e diminuisce di una percentuale fissata. In questo modo, nel momento in cui l’agente acquisisce più informazioni riguardo l’ambiente, inizia a sfruttare più frequentemente la Q-table creata. L’agente diventerà ”greedy” in termini di sfruttamento dell’ambiente una volta che avrà avuto l’opportunità di esplorarlo e imparare quanto più possibile da esso.
Per determinare se l’agente sceglierà l’exploration o l’exploitation ad ogni tempo t, generiamo un numero random tra 0 ed 1. Se questo numero è più grande di e, allora l’agente sceglierà l’azione successiva mediante exploitation ovvero sceglierà l’azione in base al più alto Q-value per lo stato corrente dalla Q-table. Altrimenti, l’azione successiva sarà scelta mediante exploration, ovvero l’azione sarà scelta in modo casuale per esplorare l’ambiente.

```
1. if random num > e:
2. # choose action via exploitation
3. else:
4. # choose action via exploration
```

5.7 Scegliere un’azione

A questo punto, la prima azione sarà scelta stocasticamente attraverso l’esplorazione dal momento che il nostro exploration rate è impostato ad 1. Questo sta a significare che, con il 100% di probabilità, l’agente esplorerà l’ambiente durante il primo episodio del gioco piuttosto che sfruttarlo. Dopo che l’agente compie un’azione, osserva il prossimo stato, osserva il reward ottenuto dall’azione

corrente e aggiorna il Q-value nella Q-table per l'azione che ha intrapreso dallo stato precedente.
Nel caso del Frozen Lake, viene fornito un reward all'agente solo nel momento in cui egli raggiunge il suo goal.

5.8 Aggiornare il Q-value

Per aggiornare il Q-value per la coppia stato-azione usiamo l'equazione di Bellman accennata precedentemente:

$$q_*(s,a) = E[R_{t+1} + \gamma \max_{a'} q_*(s',a')]$$

Data la coppia stato-azione (s, a), l'obiettivo è rendere il Q-value q(s, a) il più possibile vicino alla parte destra dell'equazione così che il Q-value eventualmente possa convergere al valore di q∗(s, a).
Per avere la convergenza, ogni volta che incontriamo una coppia (s, a):

- Si calcola la perdita tra il Q-value e il Q-value ottimale per (s,a)
- Si aggiorna il Q-value al fine di diminuire la perdita.

$$q_*(s,a) - q(s,a) = loss$$

$$E[R_{t+1} + \gamma \max_{a'} q_*(s',a')] - E[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}] = loss$$

Per vedere come aggiornare il Q-value, bisogna prima introdurre l'idea di learning rate.

5.9 Il learning rate

Il learning rate è un numero tra 0 e 1 che può essere pensato come quanto velocemente l'agente abbandona il Q-value precedente nella Q-table (data una coppia stato-azione) per un nuovo Q-value.
Supponiamo di avere un Q-value nella Q-table per alcune coppie stato-azione arbitrarie che l'agente ha avuto modo di apprendere allo step precedente. Se l'agente apprendesse le stesse coppie stato-azione al passo successivo quando ha imparato di più circa l'ambiente, il Q-value avrà bisogno di essere aggiornato per riflettere il cambiamento nelle aspettative che l'agente possiede in quel momento per il futuro.
Non vogliamo solo sovrascrivere il vecchio Q-value ma, piuttosto, vogliamo usare il learning rate come uno strumento per determinare quanta informazione mantenere circa il precedente Q-value data la coppia stato-azione, contro il nuovo Q-value calcolato per la stessa coppia stato-azione al passo successivo. Denotiamo il learning rate con il simbolo α e assegneremo arbitrariamente α = 0.7.
Più è alto il valore di learning rate, più velocemente l'agente adotterà il nuovo Q-value. Per esempio, se il learning rate è 1, la stima per il Q-value, data la coppia stato-azione, sarebbe direttamente il nuovo Q-value calcolato e non verrebbe considerato il precedente Q-value che è stato calcolato, data la coppia stato-azione al passo precedente.

5.10 Calcolare il nuovo Q-value

La formula per calcolare il nuovo Q-value per una coppia stato-azione (s, a) al tempo t è:

$$q^{new}(s,a) = (1-\alpha)\ \underbrace{q(s,a)}_{old\ value} + \alpha\overbrace{(R_{t+1}) + \gamma \max_{a'} q(s',a')}^{new\ value}$$

$$= (1-0.7)(0) + 0.7(-1 + 0.99(\max_{a'}\ q(s',a')))$$

Così il nuovo Q-value è uguale alla somma pesata del vecchio Q-value e il Q-value ottimo in base al learning value. Il vecchio valore nel nostro caso è 0 perchè è la prima volta che l'agente sta incontrando questa particolare coppia stato-azione e moltiplichiamo questo vecchio valore per $(1-\alpha)$.

Il nostro valore appreso è il reward che l'agente riceve muovendosi dallo stato iniziale più la stima 0 0scontata dell'ottimo futuro Q-value per la prossima coppia stato-azione (s', a') al tempo $t+1$. Questo valore appreso è poi moltiplicato per il nostro learning rate. Supponendo che il discount rate $\gamma = 0.99$ avremmo:

$$q^{new}(s,a) = (1-\alpha)\ \underbrace{q(s,a)}_{old\ value} + \alpha\overbrace{(R_{t+1}) + \gamma \max_{a'} q(s',a')}^{new\ value}$$

$$= (1-0.7)(0) + 0.7(-1 + 0.99(\max_{a'}\ q(s',a')))$$

Focalizziamoci sul termine $\max_{a'}\ q(s', a')$. Dal momento che tutti i Q-values sono inizializzati a 0 nella Q-table abbiamo:

$$\max_{a'}\ q(s',a') = \max(q(empty\ 6, left), q(empty\ 6, right), q(empty\ 6, up), q(empty\ 6, down))$$

$$\max(0,0,0,0)$$

$$= 0$$

Adesso possiamo sostituire il valore 0 in $\max q(s', a')$nella precedente equazione per risolvere $q^{new}(s,a)$.

$$q^{new}(s,a) = (1-\alpha)\ \underbrace{q(s,a)}_{old\ value} + \alpha\overbrace{(R_{t+1}) + \gamma \max_{a'} q(s',a')}^{new\ value}$$

$$= (1-0.7)(0) + 0.7(-1 + 0.99(\max_{a'}\ q(s',a')))$$

$$= (1-0.7)(0) + 0.7(-1 + 0.99(0))$$

$$= 0 + 0.7(-1)$$

$$= -0.7$$

Adesso prenderemo questo nuovo Q-value appena calcolato e lo memorizzeremo nella nostra Q-table per questa particolare coppia stato-azione. Questo è tutto ciò che serve per un singolo step. Lo stesso processo avverrà ogni step successivo finchè l'episodio non termina. Quando la Q-function converge alla Q-function ottimale, avremo la nostra ottima policy.

5.11 Numero massimo di step

Possiamo anche specificare un numero massimo di step che il nostro agente esegue prima che l'episodio termini anche se non raggiunge il goal. Per esempio si potrebbe definire una condizione per cui se l'agente non ha raggiunto la terminazione entro 100 step allora il gioco terminerà al 100 step.

6 Implementazione

Per risolvere il gioco Frozen-Lake abbiamo utilizzato diversi approcci. Abbiamo considerato entrambe le dimensioni dell'ambiente di Frozen-Lake, utilizzando in entrambi i casi due approcci.

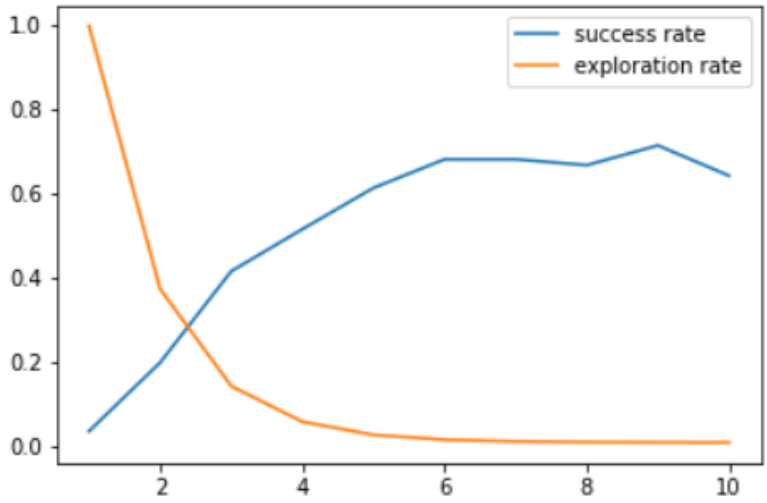
6.1 Frozen-Lake 4x4

I due approcci ottengono risultati differenti. Nel caso dell'algoritmo Q-Learning Genetico si riescono ad ottenere delle Q-Table con tassi di successo più alti del 5-6%. Nonostante ciò l'algoritmo di Q-Learning Genetico è computazionalmente più oneroso.

6.1.1 Q-Learning base

In questo approccio abbiamo utilizzato un algoritmo di Q-Learning base con i seguenti parametri:

- num_episodes = 10000
- max_steps_per_episode = 100
- learning_rate = 0.1
- discount_rate = 0.99
- exploration_rate = 1
- max_exploration_rate = 1
- min_exploration_rate = 0.01
- exploration_decay_rate = 0.001



Relazione tra calo dell'exploration rate e il tasso di successo. Sulle ascisse il numero di episodi/100

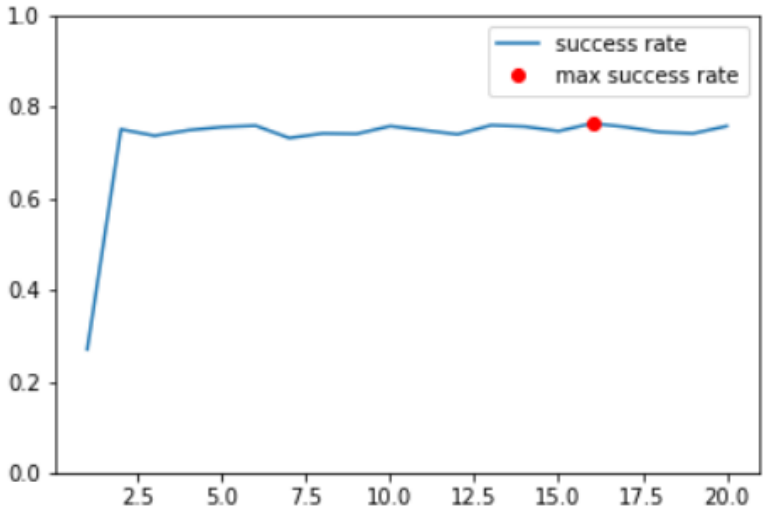
Dopo 10.000 episodi l'algoritmo raggiunge una percentuale di successo vicina al 70%. Superato questo numero di episodi, la percentuale di successo oscilla ma rimane sostanzialmente stabile intorno a quel numero. Questo è dovuto alla casualità dei movimenti, stesso motivo per cui è difficile raggiungere una percentuale di successo vicina al 100%.

6.1.2 Q-Learning con algoritmo genetico

Questo approccio prevede che ci siano 10 agenti, ognuno con una sua Q-table. Le Q-table vengono inizializzate casualmente e ogni agente effettua 1.000 episodi di Q-Learning base. Vengono quindi valutate le Q-table su 1.000 episodi di test. Le Q-table che raggiungono un maggior tasso di successo avranno più probabilità di essere selezionate come genitori per il crossover (Stochastic Beam Search). Le 2 Q-table migliori vengono utilizzate anche nella generazione successiva. Il crossover avviene selezionando casualmente un indice di riga della Q-table. Sia i il numero casuale generato nell'intervallo [1, n], dove n è il numero di righe della Q-table. Il primo figlio avrà le righe da 1 a i del primo genitore e le righe da i ad n del secondo genitore. Con i seguenti parametri si raggiunge un tasso di successo 75-76%. I risultati sono dipendenti dalla configurazione iniziale ma generalmente l'algoritmo riesce comunque a trovare (più o meno velocemente) delle Q-Table con questo tasso di successo.

- population_size = 10
- num_generation = 20
- num_training_episodes = 1000
- num_test_episodes = 1000
- max_steps_per_episode = 1000
- learning_rate = 0.99
- exploration_rate = 1

- max_exploration_rate = 1
- min_exploration_rate = 0.01
- exploration_decay_rate = 0.001



Tasso di successo sulle ordinate, generazione sulle ascisse

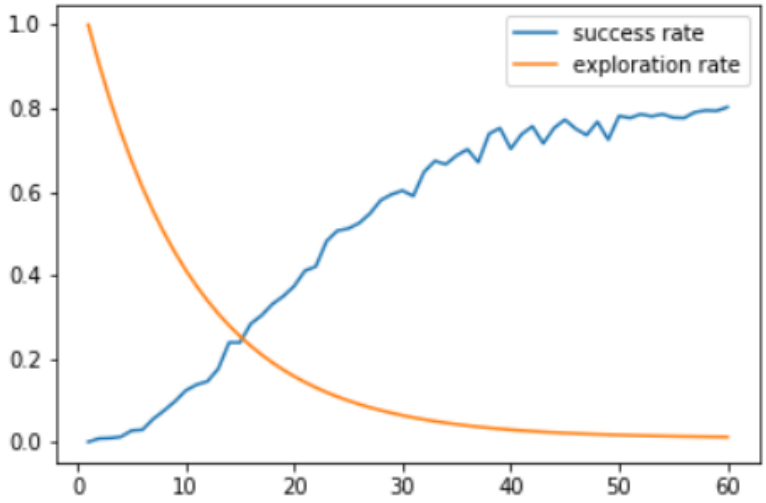
6.2 Frozen-Lake 8x8

I due approcci ottengono risultati differenti. Il Q-Learning Genetico riesce in alcune esecuzioni ad ottenere delle Q-Table con tassi di successo più alti del 4-5%. Tuttavia è molto dipendente dalla configurazione iniziale e dalla casualita, ed è computazionalmente molto più oneroso.

6.2.1 Q-Learning base

In questo caso l'algoritmo è lo stesso del caso 4x4, cambiano solo alcuni parametri:

- num_episodes = 60000
- max_steps_per_episode = 500
- learning_rate = 0.1
- discount_rate = 0.99
- exploration_rate = 1
- max_exploration_rate = 1
- min_exploration_rate = 0.01
- exploration_decay_rate = 0.0001



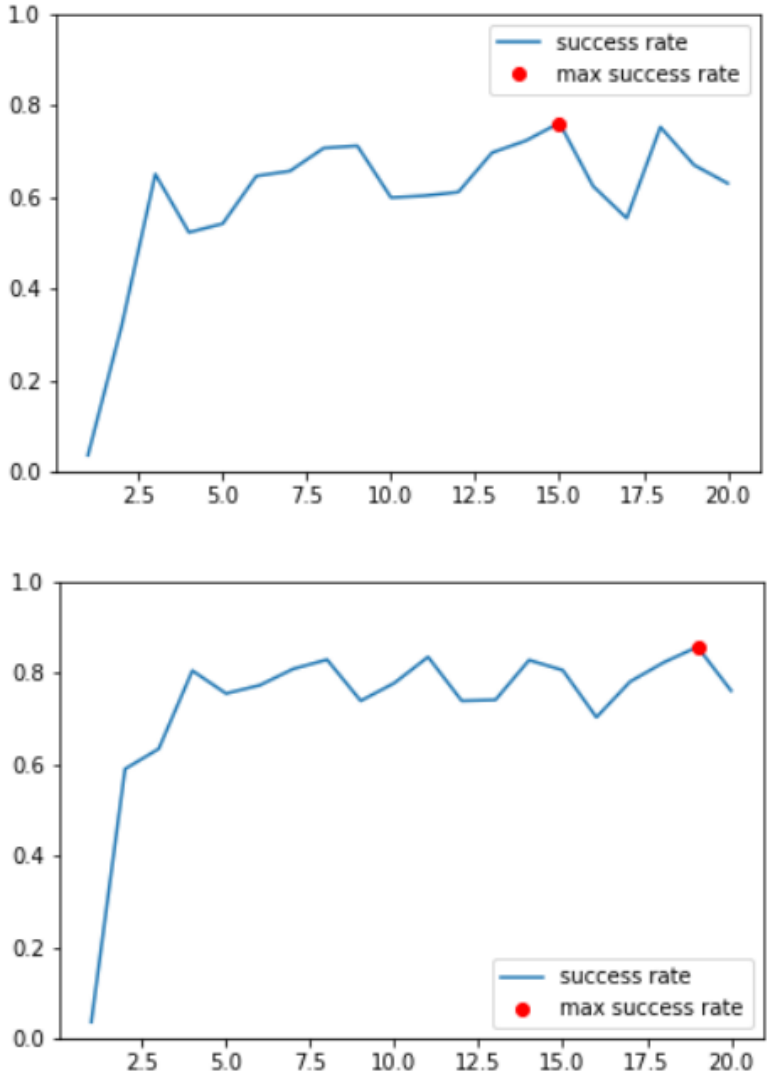
Relazione tra calo dell'exploration rate e il tasso di successo. Sulle ascisse il numero di episodi/100

Dopo 60.000 episodi l'algoritmo raggiunge un tasso di successo vicino all'80%. Superato questo numero di episodi, la percentuale di successo oscilla ma rimane sostanzialmente stabile intorno a quel numero.

6.2.2 Q-Learning con algoritmo genetico

L'algoritmo è il medesimo del caso 4x4. Cambiano solo alcuni parametri:

- num_generation = 20
- num_training_episodes = 5000
- num_test_episodes = 1000
- max_steps_per_episode = 1000
- learning_rate = 0.1
- discount_rate 0.99
- exploration_rate = 1
- max_exploration_rate = 1
- min_exploration_rate = 0.01
- exploration_decay_rate = 0.0005



Tasso di successo sulle ordinate, generazione sulle ascisse

I risultati in questo caso oscillano molto più dei casi precedenti, ma ci sono alcuni picchi intorno all'80% di tasso di successo. In diverse esecuzioni si riescono ad ottenere risultati abbastanza diversi.
E stato anche registrato un pico di 85%