**Introducere**

In acest articol voi vorbi despre modelul matematic q-learning și fundalul de bază pentru înțelegerea algoritmului.Începînd cu cel mai important lucru, adică cu facerea cunoștinței despre ce este q-learning. Cum se discifreaza q-learnig, cum se creaza un tabel-Q, cum se face realizarea de actualizări, care sunt cei 3 pașii de bază, ce este gamma, maxim, reconpensa și rata de învătare și cum ele se aplică în program.

**Ce este q-learning?**

Q-learning este o consolidare learni politicii off n g algoritm care caută să găsească cea mai bună acțiune pentru care ia dat starea actuală. Este considerat în afara politicii, deoarece funcția q-learning învață din acțiunile care sunt în afara politicii actuale, cum ar fi luarea de acțiuni aleatorii și, prin urmare, nu este necesară o politică. Mai exact,q-learning caută să învețe o politică care maximizează recompensa totală.

**Ce este „Q”?**

„Q” din q-learning înseamnă calitate. Calitatea în acest caz reprezintă cât de utilă este o anumită acțiune în obținerea unei recompense viitoare.

**Creați un tabel Q**

Când se realizează q-learning, creăm ceea ce se numește un *q-tabel* sau o matrice care urmează forma [state, action]și ne inițializam valorile la zero. Apoi ne actualizăm și stocăm *valorile q* după un episod. Acest tabel q devine un tabel de referință pentru agentul nostru pentru a selecta cea mai bună acțiune pe baza valorii q.

import numpy ca np# Inițializați valorile tabelului q la 0Q = np.zeros((state\_size, action\_size))

**Q-learning și realizarea de actualizări**

Următorul pas este pur și simplu ca agentul să interacționeze cu mediul și să facă actualizări ale perechilor de acțiuni de stare din tabelul nostru q  Q[state, action].

*Acțiune: Explorează sau exploatează*

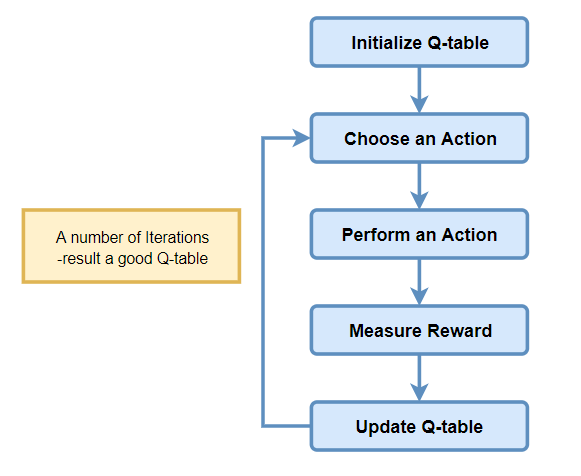
Un agent interacționează cu mediul în 1 din 2 moduri. Prima este să folosiți tabelul q ca referință și să vizualizați toate acțiunile posibile pentru o anumită stare. Agentul selectează apoi acțiunea pe baza valorii maxime a acelor acțiuni. Acest lucru este cunoscut sub numele de ***exploatare,*** deoarece folosim informațiile pe care le avem la dispoziție pentru a lua o decizie.

A doua modalitate de a acţiona este să acţionezi la întâmplare. Aceasta se numește ***explorare*** . În loc să selectăm acțiuni pe baza recompensei maxime viitoare, selectăm o acțiune la întâmplare. Acționarea aleatorie este importantă deoarece permite agentului să exploreze și să descopere noi stări care altfel ar putea să nu fie selectate în timpul procesului de exploatare. Puteți echilibra explorarea/exploatarea folosind epsilon ( *ε* ) și setând valoarea cât de des doriți să explorați vs. exploatare. Iată un cod brut care va depinde de modul în care sunt configurate starea și spațiul de acțiune.

Import aleatoriu# Setați procentul pe care doriți să îl explorați   
epsilon = 0,2if random.uniform(0, 1) < epsilon:   
 """   
 Explorare: selectați o acțiune aleatorie """   
else:   
 """   
 Exploat: selectați acțiunea cu valoarea maximă (recompensă viitoare) """

*Actualizarea tabelului q*

Actualizările au loc după fiecare pas sau acțiune și se termină când se termină un episod. Efectuat în acest caz înseamnă atingerea unui punct terminal de către agent. O stare terminală, de exemplu, poate fi ceva precum aterizarea pe o pagină de plată, atingerea sfârșitului unui joc, îndeplinirea unui obiectiv dorit etc. Agentul nu va învăța prea multe după un singur episod, dar în cele din urmă cu suficientă explorare (pași și episoade) va converge și va învăța valorile q optime sau q-stea ( Q∗).



**Iată cei 3 pași de bază:**

1. Agentul începe într-o stare (s1) efectuează o acțiune (a1) și primește o recompensă (r1)
2. Agentul selectează acțiunea făcând referire la tabelul Q cu cea mai mare valoare (max) **SAU** la întâmplare (epsilon, ε)
3. Actualizați valorile q

Iată regula de bază de actualizare pentru q-learning:

# Actualizați valorile qQ[stare, acțiune] = Q[stare, acțiune] + lr \* (recompensă + gamma \* np.max(Q[stare\_nouă, :]) — Q[stare, acțiune])

În actualizarea de mai sus există câteva variabile pe care nu le-am menționat încă. Ceea ce se întâmplă aici este că ne ajustăm valorile q în funcție de diferența dintre noile valori actualizate și valorile vechi. Reducem noile valori folosind gamma și ne ajustam dimensiunea pasului folosind rata de învățare (lr). Mai jos sunt câteva referințe.

**Rata de învățare:** lr sau rata de învățare, denumită adesea alfa sau α, poate fi definită pur și simplu ca cât de mult acceptați noua valoare față de valoarea veche. Mai sus luăm diferența dintre nou și vechi și apoi înmulțim acea valoare cu rata de învățare. Această valoare este apoi adăugată la valoarea q anterioară, ceea ce o mută în esență în direcția ultimei noastre actualizări.

**Gamma:** gamma sau γ este un factor de reducere. Este folosit pentru a echilibra recompensa imediată și cea viitoare. Din regula noastră de actualizare de mai sus puteți vedea că aplicăm reducerea recompensei viitoare. De obicei, această valoare poate varia între 0,8 și 0,99.

**Recompensa:** reward este valoarea primită după finalizarea unei anumite acțiuni într-o anumită stare. O recompensă poate avea loc la orice pas de timp dat sau numai la pasul de timp terminal.

**Max:** np.max() folosește biblioteca numpy și ia maximul recompensei viitoare și îl aplică recompensei pentru starea actuală. Ceea ce face aceasta este un impact asupra acțiunii curente prin posibila recompensă viitoare. Aceasta este frumusețea q-learning-ului. Alocam recompense viitoare pentru acțiunile curente pentru a ajuta agentul să aleagă cea mai mare acțiune de rentabilitate în orice stare dată.

**Concluzie**

Am discutat că q-learning este un algoritm de învățare de întărire în afara politicii. Arătăm regula de bază de actualizare pentru q-learning folosind o sintaxă python de bază și am revizuit intrările necesare pentru algoritm. Am învățat că q-learning utilizează recompense viitoare pentru a influența acțiunea curentă dată de o stare și, prin urmare, ajută agentul să aleagă cele mai bune acțiuni care maximizează recompensa totală.

**Resurse:**

1. Great RL and q-learning example using the [OpenAI Gym taxi](https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/)environment
2. [Reinforcement Learning: An Introduction](http://www.incompleteideas.net/book/RLbook2018trimmed.pdf) (free book by Sutton)
3. Quora [Q-learning](https://www.quora.com/How-does-Q-learning-work-1)
4. Wikipedia [Q-learning](https://en.wikipedia.org/wiki/Q-learning)
5. David Silver’s [lectures on RL](http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html)
6. <https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/>
7. <https://courses.cs.ut.ee/MTAT.03.292/2014_spring/uploads/Main/Q-learning.pdf>
8. <https://blog.dominodatalab.com/deep-reinforcement-learning/>