5.5.1 Q Learning & Deep Q Learning

Tags

Recap

Monte Carlo vs TD Method

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Monte Carlo update:

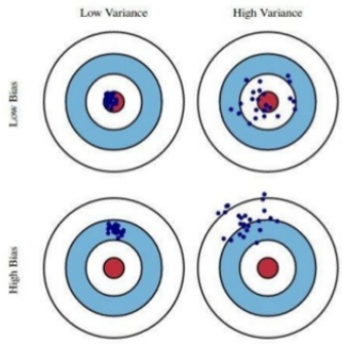
$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

- Unbiased

The target is true return of the sample.

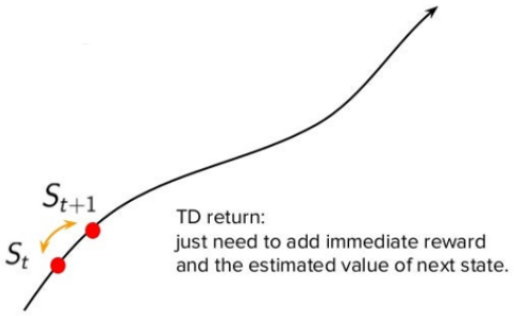
- High variance

The targets among samples are very different because they are depend on whole trajectory.



TD method:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]\text{c}$$



TD method:

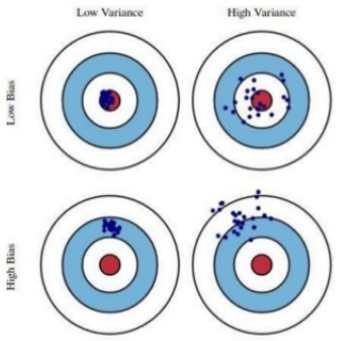
$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]\text{c}$$

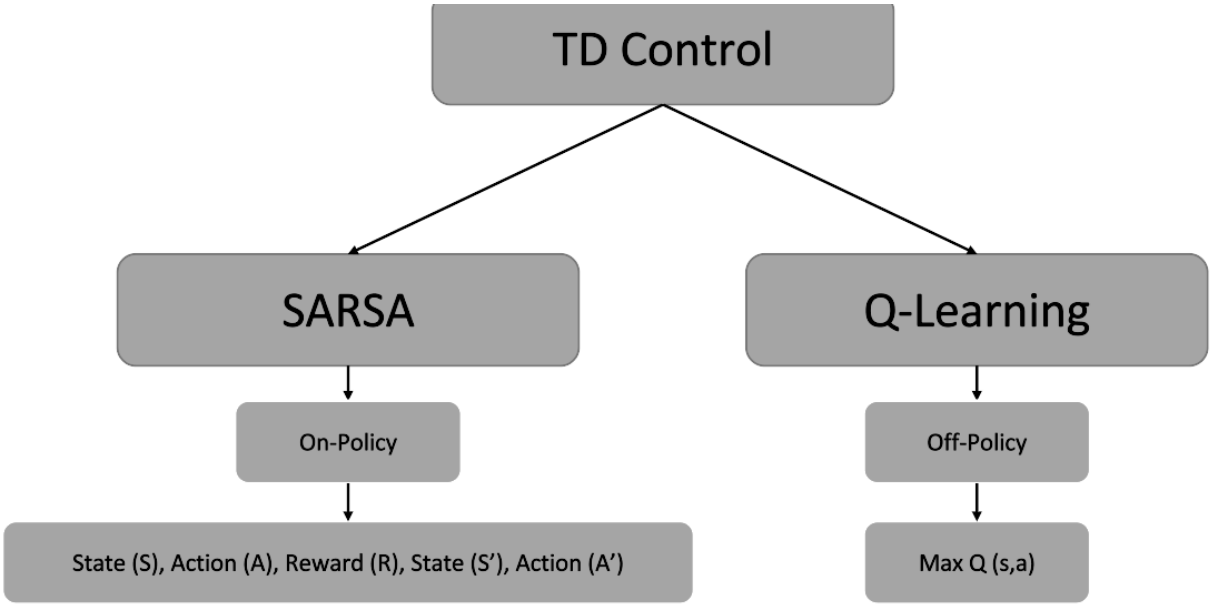
- Biased

The target is also an estimated value (because of V(s)).

- Low variance

The targets are slightly difference, because it just take one-step.





Q Learning

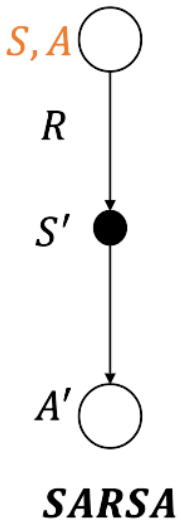
SARSA vs Q Learning

- On-policy: Sample policy sama dengan learning policy (target policy)
  Contoh: Sarsa dan Policy Gradient
- Off-policy: Sample policy berbeda dengan learning policy (target policy)
  Contoh: Q Learning dan Deep Q Network (DQN)

SARSA On Plicy TD Control

- Terinspirasi dari policy iteration
- Mengganti value function (V_π) dengan action-value function
- On Policy
- Fokus pada state-action (S, A)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_T) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

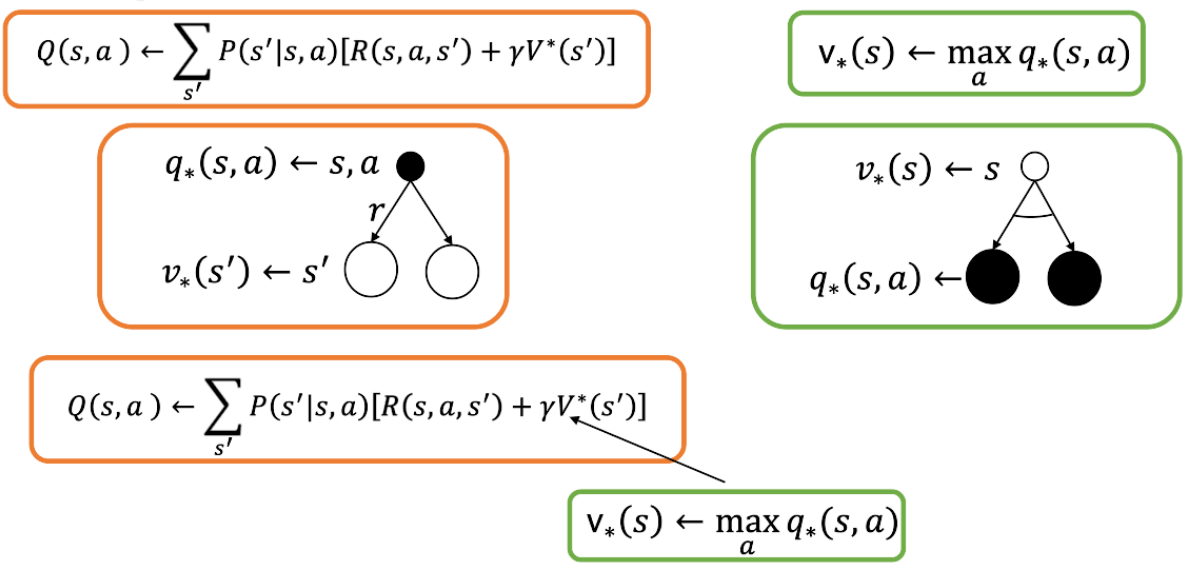


Q-Learning Off Policy TD Control

Q-Learning merupakan pengembangan RL yang menggunakan Q-values (disebut juga action-values) untuk meningkatkan kemampuan agent belajar agent berulang-ulang.

Konsep dasar Q-Learning:

- Terinspirasi dari value iteration



$$Q(s,a) \leftarrow \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q(s',a')\right]$$

- Q Learning update

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

- Sample an action
- Observe the reward and the next state
- Take the action with the highest Q (Max Q)

Action dari setiap step dapat dihitung untuk menemukan action terbaik (best action). Untuk keperluan ini digunakan Q-Table.

Algoritma Q-Learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Parameter algoritma: step size $\alpha \in (0,1]$, small $\epsilon > 0$
Inisialisasi $Q(s,a)$, untuk semua $s = S^+, a \in \mathrm{A}(s)$, arbitrary except that

$Q(terminal, .) = 0$

Loop untuk setiap episode:

Inisialisasi S

Loop untuk setiap step dari episode:

Pilih A dari S menggunakan policy derived from $Q(e.g., \epsilon - greedy)$

Ambil action A, observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma max_a q(S', a) - Q(S, A)]$

$S \leftarrow S'$

hingga S is terminate

Algoritma Q Learning secara sederhana:

1. Tentukan current state = initial state
2. Dari current state, cari dengan nilai Q terbesar
3. Tentukan current state = next state.
4. Ulangi langkah 2 dan 3 hingga current state = goal state.

Algoritma Q Learning (pseudocode)

Algorithm: Q Learning - Learn function

Require.

States $X = \{1, ..., n_x\}$

Action $\mathrm{A} = \{1, ..., n_\alpha\}, A : X \Rightarrow \mathrm{A}$

Reard function $R : X \times \mathrm{A} \to \mathbb{R}$

Black - box (probabilitastic) transition function $T : X \times \alpha \to X$

Learing rate $\alpha \in [0, 1], a = 0.1$

Discounting factor $\gamma \in [0, 1]$

Procedure Q - Learning ($X, \mathrm{A}, \mathbb{R}, T, \alpha, \gamma$)

Inisialisasi $Q : X \times \mathrm{A} \to \mathbb{R}$ *arbitrarily*

while Q is not converged do

Start in state $s \in X$

while s *is not terminal do*

Calculating π *according to Q and exploration strategy* $(e.g. \pi(x) \leftarrow argmax_a Q(x,a))$

$\alpha \leftarrow \pi(s)$

$r \leftarrow R(s,a)$

$s' \leftarrow T(s,a)$

$Q(s',a) \leftarrow (1-\alpha).Q(s,a) + a.(r + \gamma.max_a Q(s',a'))$

$s \leftarrow s'$

end

end

return Q

Q Learning

Objektif dari Q Learning adalah mencari policy yang optimal berdasarkan in the sense that the expected value of the total reward over all successive steps is the maximum achievable. Atau dengan kata lain, tujuan dari Q-Learning adalah mencari policy yang optimal dan mencari optimal Q-Values untuk setiap pasangan state-action.

Terminologi

Policy (π) : Adalah sebuah fungsi yg memetakan setiap state ke action. Sehingga dengan mengikuti policy, agent bisa memilih action apa yg akan dia ambil pada state tertentu.

Reward (r) : feedback atau umpan balik untuk agent. Reward dapat bernilai positif (berupa hadiah) atau negatif (berupa hukuman) dan juga nol (tidak ada tindakan apapun terhadap agent).

Episodes: Ketika agent berakhir pada terminating state dan tidak bisa mengambil action apapun pada proses learning.

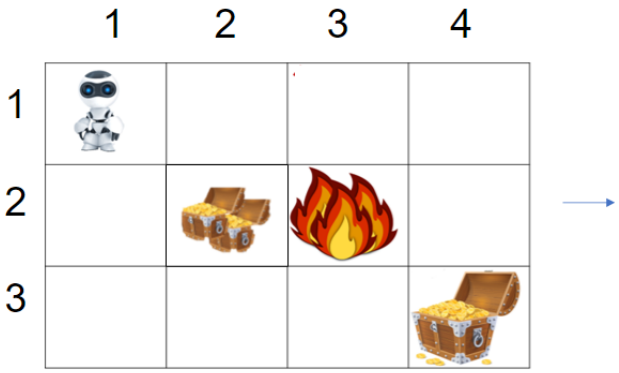
Q Function : adalah fungsi yang digunakan dalam Q-Learning untuk mencari Q-Value untuk setiap pasangan state dan action dan untuk menentukan apakah sebuah action

akan baik jika dilakukan pada state tertentu.

Lalu bagaimana kita menemukan Q-Value yang terbaik menggunakan Q-Function? Kita gunakan persamaan berikut :

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

Q-Table : adalah sebuah table yang digunakan untuk menyimpan Q-Value untuk setiap pasangan state dan action. Horizontal axis dari table ini merepresentasikan kumpulan action, dan vertical axis merepresentasikan kumpulan state yang ada. Jadi, dimensi dari table akan bergantung terhadap jumlah actions dan jumlah states.



Actions

States

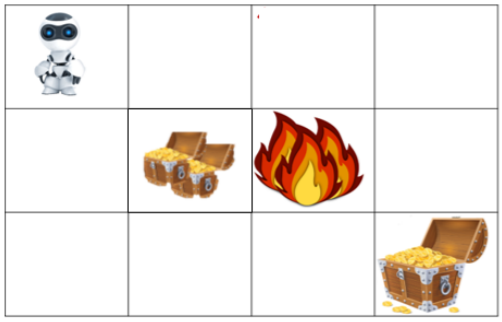
	Kiri	Kanan	Atas	Bawah
(1,1)	Q-Value	Q-Value	Q-Value	Q-Value
(1,2)	Q-Value	Q-Value	Q-Value	Q-Value
(1,3)	Q-Value	Q-Value	Q-Value	Q-Value
(1,4)	Q-Value	Q-Value	Q-Value	Q-Value
(2,1)	Q-Value	Q-Value	Q-Value	Q-Value
(2,2)	Q-Value	Q-Value	Q-Value	Q-Value
(2,3)	Q-Value	Q-Value	Q-Value	Q-Value
(2,4)	Q-Value	Q-Value	Q-Value	Q-Value
(3,1)	Q-Value	Q-Value	Q-Value	Q-Value
(3,2)	Q-Value	Q-Value	Q-Value	Q-Value
(3,3)	Q-Value	Q-Value	Q-Value	Q-Value
(3,4)	Q-Value	Q-Value	Q-Value	Q-Value

How Q learning work?

Studi Kasus

Studi kasus kali ini kita akan membantu robot
Orbit untuk menemukan harta karun yang paling banyak dengan step yg sedikit dan menghindari supaya tidak terkena api azab.

1. **Definisi Reward dan Q Tabel**



State	Reward
Empty	-1
Fire	-10
1 Harta Karun	+10
2 Harka Karun	+25

Actions

States

	Kiri	Kanan	Atas	Bawah
(1,1)	0	0	0	0
(1,2)	0	0	0	0
(1,3)	0	0	0	0
(1,4)	0	0	0	0
(2,1)	0	0	0	0
(2,2)	0	0	0	0
(2,3)	0	0	0	0
(2,4)	0	0	0	0
(3,1)	0	0	0	0
(3,2)	0	0	0	0
(3,3)	0	0	0	0
(3,4)	0	0	0	0

2. **Chose Action (Eksplorasi vs Eksploitasi)**

   Eksplorasi adalah suatu metode pemilihan action dimana agent akan melakukan pemilihan action secara random dengan tujuan ia bisa mengetahui informasi tentang environment secara mendalam.

   Sedangkan **Eksploitasi** adalah sebuah metode pemilihan action dengan memilih action yang mempunyai return (dalam hal ini Q-Value) paling besar.

   Pada dasarnya dalam algoritma Q Learning, agent akan memilih action berdasarkan Q-Tabel. Agent akan memilih action yang mempunyai Q-Value paling besar berdasarkan state saat ini. Tapi hal tersebut akan terdengar aneh untuk langkah pertama, bukan? Karena pada langkah pertama semua Q-Value yang ada pada Q-Tabel bernilai 0, jadi gimana cara agent memilih action?

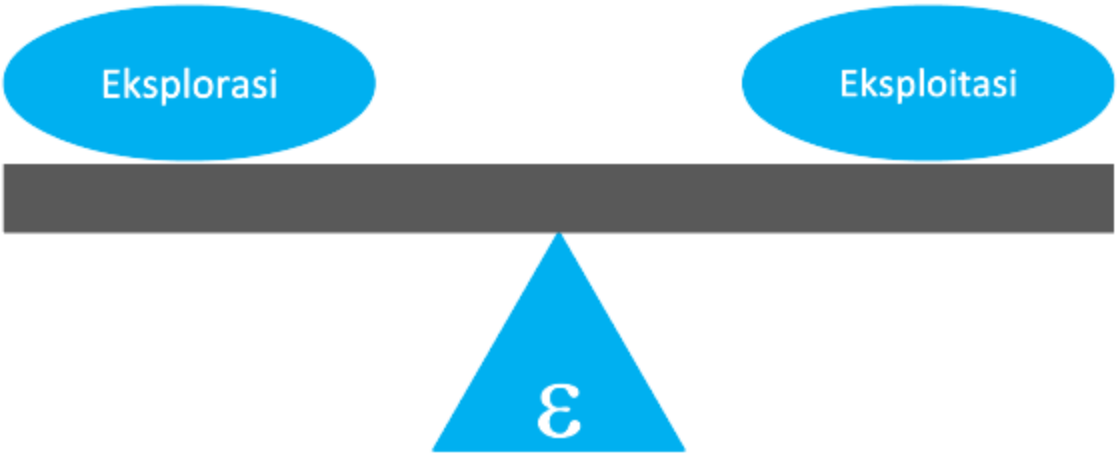
   Mari kita bayangkan environment kita adalah studi kasus pada kali ini.

   Bagaimana kalau agent (robot) menemukan terlebih dahulu satu harga karun, lalu kita menggunakan metode eksploitasi dalam memilih action? Maka yang terjadi adalah agent akan cenderung memilih action yang akan menuntuk dia ke satu harta karun dibandingkan dua hartakaurn.

   Lalu apakah metode eksplorasi adalah yang terbaik? oh tentu tidak juga. Jika kita menggunakan metode eksplorasi (memilih action secara random) secara terus menerus maka kita akan kesulitan untuk mencapai solusi yang optimum.

   solusi terbaiknya adalah menggunakan metode **epsilon-greedy exploration.**

   Epsilon-greedy exploration digunakan untuk menyeimbangkan antara ekplorasi dan ekploitasi menggunakan nilai epsilon. nilai epsilon bisa disebut dengan rate eksplorasi



Pertama set nilai $\epsilon = 1$, itu artinya pada episode-episode awal kita akan menggunakan metode eksplorasi dalam memilih action dengan tujuan agar agent dapat mengenali environtmentnya dengan baik.

Lalu dengan seiring berjalannya waktu nilai ϵ akan kita kurangi dengan rate tertentu, sehingga agent akan cenderung memilih action menggunakan metode eksploitasi. Lalu, untuk memberi tahu agent kapan harus pakai eksplorasi dan kapan harus eksploitasi, kita akan generate nomor acak dari 0 sampai 1.

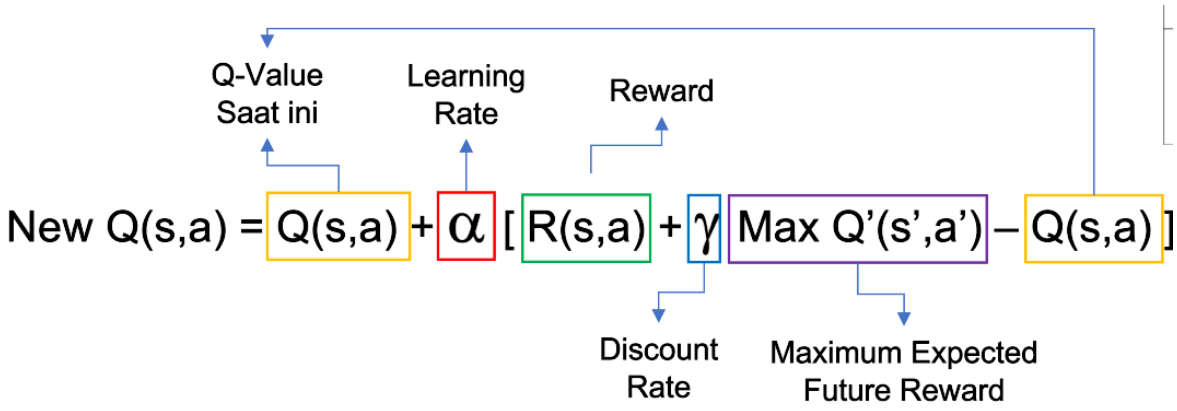
Kembali ke langkah kedua yaitu pemilihan action, maka sesuai dengan metode **epsilon-greedy exploration**, kita akan set nilai $\epsilon = 1$ dan untuk awal kita akan memilih action secara random. Kita misalkan agent memilih action ke bawah.

3. **Perform Action and Get Reward**

$$Reward = -1$$

Setelah eksekusi action dan mendapatkan reward, saatnya kita mengupdate Q-Tabel dengan mengganti Q-Value yang lama dengan Q-Value yang baru, untuk mengupdate Q-Tabel dengan menghitung Q-Value yang baru kita dapat menggunakan formula berikut.

4. **Update Q-Tabel**



Kita akan coba update Q-Tabel untuk agent bergerak ke bawah, dengan kita

$$\text{New } Q(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma MaxQ'(s',a') - Q(s,a)]$$

$$MaxQ'(s',a') = Max(q((2,1),kiri), q((2,1),kanan), q((2,1),atas), q((2,1),bawah))$$

$$MaxQ'(s',a') = Max(0,0,0,0)$$

$$MaxQ'(s',a') = 0$$

	Kiri	Kanan	Atas	Bawah
(1,1)	0	0	0	0
(1,2)	0	0	0	0
(1,3)	0	0	0	0
(1,4)	0	0	0	0
(2,1)	0	0	0	0
(2,2)	0	0	0	0
(2,3)	0	0	0	0
(2,4)	0	0	0	0
(3,1)	0	0	0	0
(3,2)	0	0	0	0
(3,3)	0	0	0	0
(3,4)	0	0	0	0

Kita akan coba update Q-Tabel untuk agent bergerak ke bawah, dengan kita definisikan $\gamma = 0.99$ dan $\alpha = 0.7$

$MaxQ'(s', a') = 0$

New $Q(s, a) = Q(s, a) + \alpha[\mathbb{R}(s, a) + \gamma MaxQ'(s', a') - Q(s, a)]$

New $Q((1,1), bawah) = 0 + 0.7[-1 + 0.99(0) - 0]$

New $Q((1,1), bawah) = 0 + 0.7[-1]$

New $Q((1,1), bawah) = -0.7$

Jadi, kita sudah melakukan kalkulasi untuk Q-Value yang baru untuk state (1,1) dan action ke bawah, dan kita juga sudah menyimpan value tersebut ke Q-Tabel.

Kita sudah selesai melakukan proses Q-Learning untuk satu step. Proses ini (langkah 2 sampai 4) akan dilakukan sampai beberapa episode. Sampai Q-tabel yang kita punya konvergen, sehingga kita menemukan policy yang paling optimal.

	Kiri	Kanan	Atas	Bawah
(1,1)	0	0	0	-0.7
(1,2)	0	0	0	0
(1,3)	0	0	0	0
(1,4)	0	0	0	0
(2,1)	0	0	0	0
(2,2)	0	0	0	0
(2,3)	0	0	0	0
(2,4)	0	0	0	0
(3,1)	0	0	0	0
(3,2)	0	0	0	0
(3,3)	0	0	0	0
(3,4)	0	0	0	0

Limitation of Q Learning

Pada game robot karun, environmetn kita sangat simpel sekali dengan hanya memiliki 12 states dan 4 action. Artinya kita hanya akan punya Q-Tabel sebasar 12x4 atau ada 48 Q-Values yang harus diupdate dalam setiap iterasi.

Bisa kita bayangkan kalau kita punya environment dengan ukuran states action yang besar, seperti Dota 2 contohnya. State dan action yang ada pada game tersebut pasti akan besar sekali, hal tersebut akan membuat ukuran Q-Table kita akan menjadi besar sekali.

Lalu bagaimana kalau kita ingin bereksperimen degnan environment yang lebih besar dari game karun?

Deep Q Learning

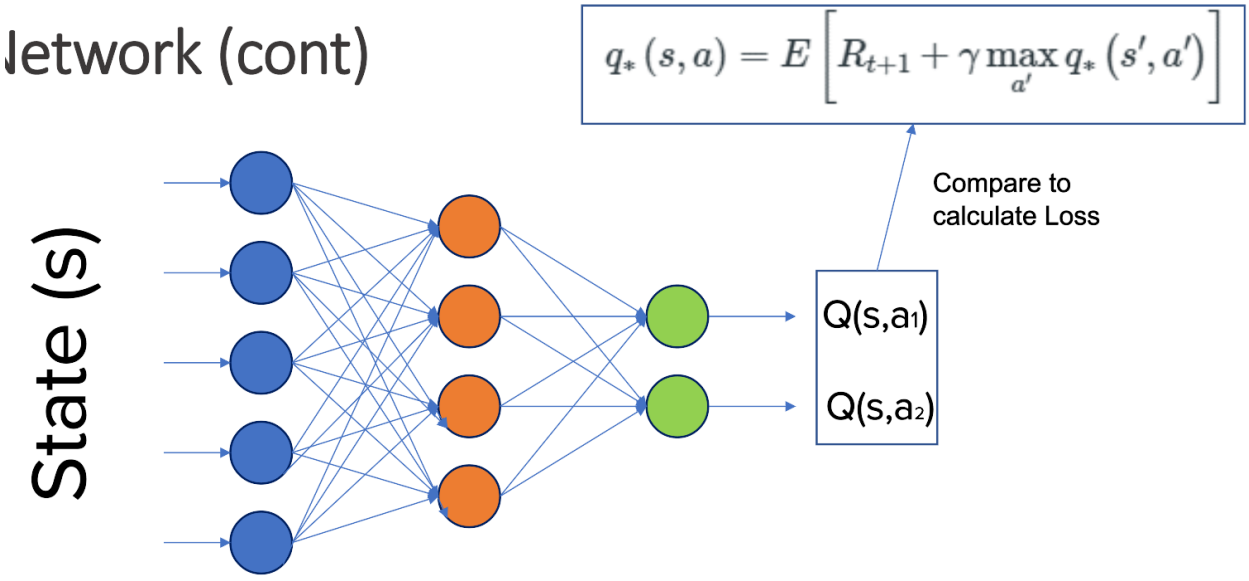
Pada deep Q learning kita akan menggantikan Q-tabel menggunakan sebuah neural network yang biasa disebut dengan Deep Q Network atau DQN.

Deep Q Network

Deep Q Network adalah sebuah NN yang menerima states yang diberikan oleh environment sebagai input, lalu DQN akan menghasilkan output estimasi Q-Values pada setiap actions yang dapat diambil pada state tersebut. Tujuan dari NN ini adalah untuk menghasilkan aproksimasi Q function yang optimal. Seperti yang kita tahu pada materi sebelumnya kalau optimal Q function adalah seperti ini:

$$q_*(s,a) = \mathbf{E}\left[\mathbb{R}_{t+1} + \gamma max_a q_*(s',a')\right]$$

Loss pada NN ini adalah dengan membandingkan antara Q-Values dari output dengan target Q-Values yang didapatkan dari persamaan diatas.



Goal dari NN ini adalah untuk minimize loss, lalu setelah menghitung loss bobot pada network akan diupdate menggunakan stochastic gradient descent dan backpropogation seperti neural network pada umumnya.

Experience Replay dan Replay Memory

Dalam proses training DQN kita akan menggunakan sebuah teknik yang dinamakan dengan **experience replay.** Dengan experience replay, kita menyimpan experience dari agent untuk setiap time step ke dalam sebuah wadah bernama replay memory.

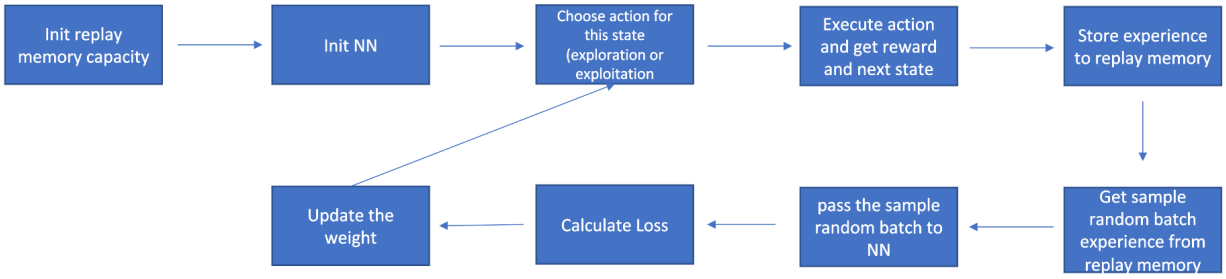
Kita dapat mempresentasikan experience dari agent untuk setiap t sebagai e_t. Pada time t, agent experience e_t didefinisikan sebagai tuple berikut:

$$e_t = (s_t, a_t, r_{t+1}, s_{t+1})$$

Tuple ini menyimpang state (S_t), action (a_t) yang diambil, reward (r_{t+1}) yang didapatkan oleh agent dan state selanjutnya (S_{t+1})

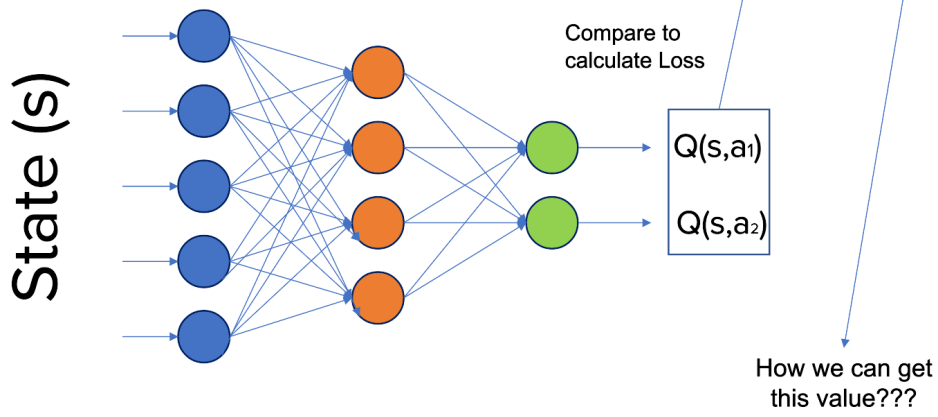
Secara teori seluruh experience agent pada setiap time step akan disimpan pada replay memory. Sebenarnya dalam praktinya, kita akan mendefinisikan besaran experience yang dapat ditampung oleh replay memory sebesar N, dan kita hanya akan menyimpan N terakhir experience dari agent.

How we Train the DQN?



How We Calculate Loss on DQN?

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

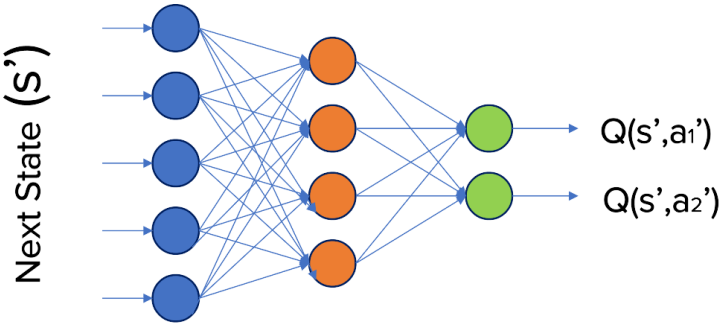


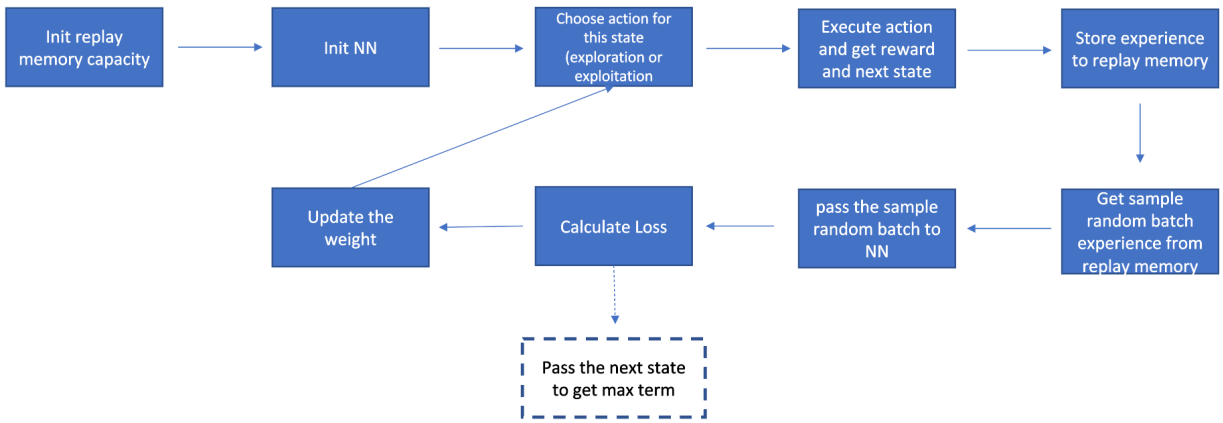
Loss = q_* - Q(s,a)

Calculate the Max Term

$$\max_{a'} q_*(s',a')$$

Pada Q Learning kita dapat mendapatkan value diatas dengan melihatnya di Q-Table, tapi itu adalah cara yg kunooo! Pada Deep Q Learning kita akan mendapatkan value tersebut dengan bantuan DQN kita





Algoritma DQN

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Rangkuman

1. Q-Learning termasuk dalam kategori Model-free RL algorithm. Menggunakan Bellman Equation untuk perhitungan dan bersifat online action-value function learning dengan exploration policy. Sehingga kita perlu belajar konsep dasar RL sebalum mempelajari Q-Learning.

2. Pada Q-Learning, agent belajar menggunakan evaluation function yang bergantung pada sekumpulan state dan sekumpulan action.
3. Pada Q-Learning dilakukan iterasi:
    a. Policy Iteration
    b. Value Iteration
4. Ditahap awal, elemen matrix Q-Tabel akan ditentukan bernilai 0 semuanya. Q-tabel akan di update sejalan dengan bertambahnya jumlah episode.
5. Secara prinsip semakin banyak iterasi episode akan semakin baik Q-Tabelnya
6. Q-tabel ini dapat dianalogikan seperti ‘memori’ bagi agent. Agent dapat menentukan path optimal berdasarkan nilai yang tertera pada Q-tabel.

Student Activity

https://bit.ly/prakRL5

- Download code yang ada di drive
- Terdapat 3 file dan 1 folder image, 3 file tersebut adalah agent_brain.py berisikan tentang algoritma QL, env.py berisikan tentang cara membuat environment yang mengambil input gambar dari folder image, dan run_agent.py yg berisikan untuk running agent.
- Untuk menjalankan program, tinggal jalankan run_agent.py saja.
- Tugas : merubah tata letak environment yg sudah ada, mulai dari posisi robot, posisi goal yang dituju, dan tata letak obstacle. Rubah di bagian env.py

References

- Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction. 2020
- Sudharsan Ravichandiran. Hands-On Reinforcement Learning with Python. 2018
- Andrea Lonza. Reinforcement Learning Algorithms with Python. 2019

- Playing Atari with Deep Reinforcement Learning by Deep Mind Technologies (paper)(https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf)
- Jie-Han Chen. Temporal-difference Learning. National Cheng Kung University, Taiwan: Slide PPT
- Human-level control through deep reinforcement learning (paper) (https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf)