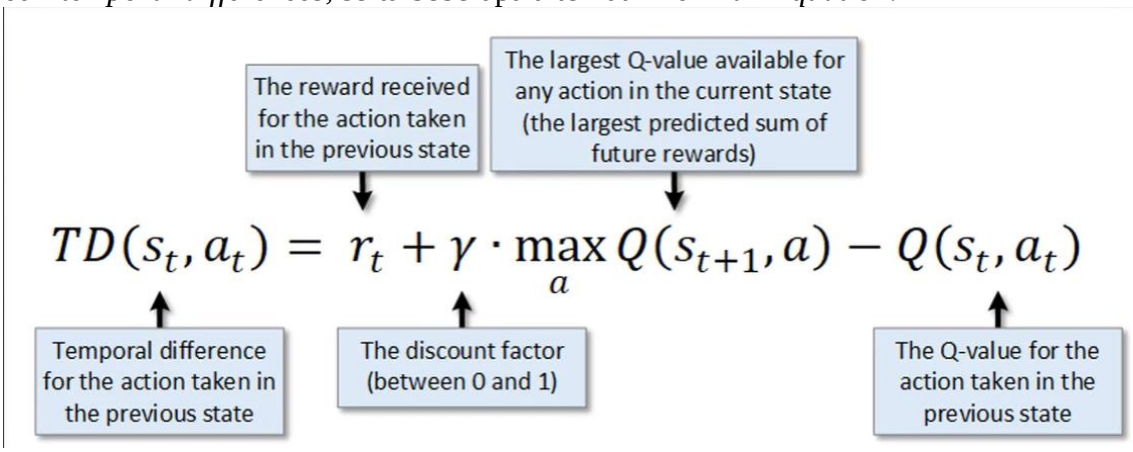
Richard Rivaldo
13519185 – K04

Q-Learning

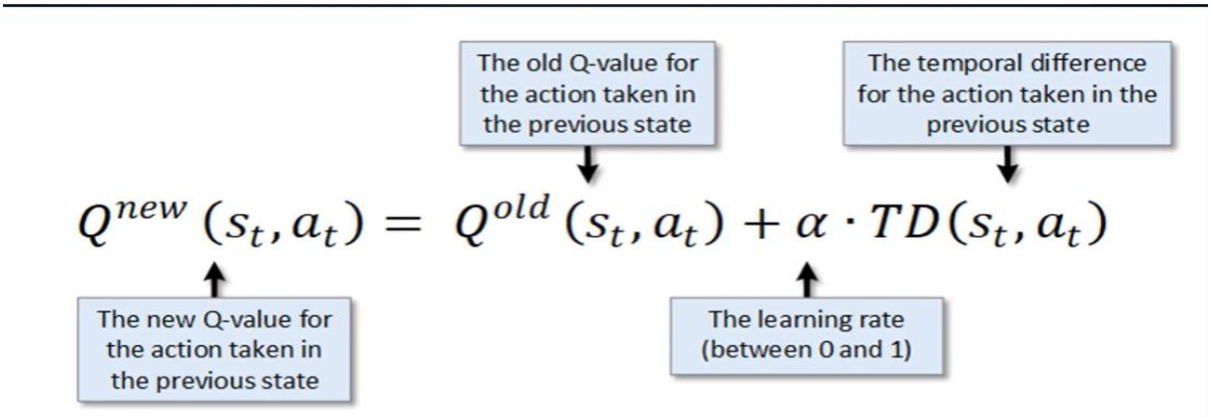
1. Jelaskan bagaimana proses dari Q-learning bekerja

Q-Learning merupakan salah satu algoritma yang termasuk ke dalam *Reinforcement Learning* dan oleh karena itu, memiliki karakteristik yang serupa dengan algoritma tersebut. Algoritma *Reinforcement Learning* merupakan kelompok lain dalam *machine learning*, selain *Supervised* dan *Unsupervised Learning*, yang utamanya melatih model yang disebut sebagai *Agent* berdasarkan kriteria tertentu. Kriteria ini utamanya didasari pada sistem *reward* dan *punishment* terhadap agen tersebut berdasarkan status tertentu yang dimilikinya.
Model *reinforcement learning* didasarkan pada lima buah prinsip dasar, yaitu sistem input dan output, sistem *rewards*, lingkungan sistem (*environment*), proses keputusan Markov, dan *training and inference mode*. Seperti model *machine learning* yang lain, model *reinforcement learning* menerima masukan berupa status dan hadiah tertentu, dengan keluaran berupa aksi tertentu. Dalam hal ini, tujuan utama dari model adalah untuk mencari *policy* optimal sehingga model bisa mengetahui aksi apa yang harus dilakukan pada status tertentu.
Rewards merupakan ukuran seberapa bermanfaat suatu aksi terhadap tujuan awal agen yang dapat berupa hadiah kumulatif atau hadiah jangka pendek. Dalam hal ini, lingkungan sistem mendefinisikan aturan yang harus dipatuhi oleh sistem ketika mencari *policy* tersebut. Setiap *environment* akan memiliki status tertentu yang tiap statusnya juga akan dikaitkan dengan *reward* tertentu. Model algoritma yang digunakan adalah *Markov Decision Process*, yaitu *framework* pengambilan keputusan pada suatu waktu *t*, status *s*, dan aksi *a* tertentu, untuk memasuki status baru pada waktu *t + 1*.
Pada mode *training*, model akan mencoba untuk memperbarui *policy* yang dimiliki sehingga lebih optimal dalam mencapai tujuan awal agen. Adapun *inference mode* merupakan usaha model dalam mengaplikasikan *policy* yang telah didapat untuk mencapai tujuan awal tersebut.
Q-Learning, juga termasuk ke dalam *model-free learning*, dalam artian bahwa model tidak mencoba untuk mencari probabilitas tertentu yang ada pada *environment* nya, tetapi berinteraksi secara langsung untuk membangun *policy* yang optimal dengan menggunakan pendekatan *trial-and-error*. Selain karakteristik di atas, pada *Q-Learning*, jumlah status dan aksi yang mungkin adalah terbatas. Adapun *Q* merupakan singkatan untuk *Quality*, dan *Q-Learning* bekerja berdasarkan *Q-Values* yang menyatakan estimasi akumulasi *rewards* di masa depan pada saat ini.
Semua *Q-Values* disimpan dan diinisialisasi dengan 0 di dalam sebuah tabel yang disebut dengan *Q-Table*. *Q-Table* adalah tabel yang terdiri dari sebuah baris untuk semua status yang mungkin, dan sebuah kolom untuk semua aksi yang mungkin untuk dilakukan agen. Sebagai contoh, dalam persoalan yang ada maka *Q-Table* berukuran 10 baris (satu baris untuk satu *tile* pada papan) dan 2 buah kolom (masing-masing untuk gerakan ke kiri dan kanan). *Q-Table* yang optimal berarti bahwa setiap *Q-Value* merepresentasikan aksi terbaik yang bisa dilakukan agen. Dengan demikian, dapat dikatakan bahwa *policy* yang dicari oleh agen ini berada dalam bentuk *Q-Table* optimal tersebut.
Cara kerja algoritma *Q-Learning* secara sederhana adalah sebagai berikut. Pertama, algoritma akan melakukan iterasi hingga mencapai semua jumlah *episode training* yang telah ditetapkan. Pada setiap *episode* ini maka *environment* akan di-*reset* sehingga kembali ke status awal. Untuk satu *episode*, maka agen akan mencoba untuk mencoba berbagai *steps* hingga jumlah *steps* nya telah mencapai *maximum steps* atau telah mencapai *terminal state* dari *environment* yang ada.

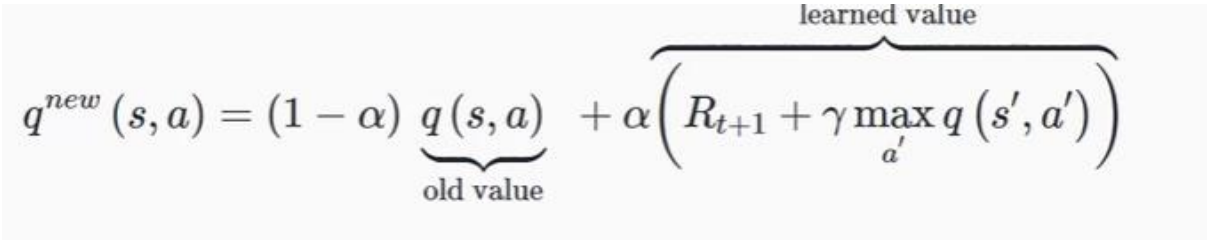
Untuk setiap *step* nya, agen akan memilih langkah atau aksi yang akan diambil dengan menggunakan *Epsilon-Greedy Strategy*. Strategi ini adalah strategi yang digunakan untuk menyeimbangkan *Exploration-Exploitation* sehingga *tradeoff* yang diberikan keduanya bisa berdampak positif untuk agen. Dengan menggunakan *tradeoff* ini, akan ditetapkan sebuah nilai *Epsilon* tertentu. Jika suatu bilangan acak didapatkan lebih tinggi dari nilai *Epsilon* pada saat tersebut, maka akan dilakukan proses *Exploitation* dengan memilih aksi dengan *Q-Value* tertinggi pada status tersebut. Jika sebaliknya, maka akan dilakukan proses *Exploration* dan aksi yang akan diambil dipilih secara acak dari semua aksi yang mungkin untuk dilakukan.
Supaya agen mengetahui aksi terbaik yang harus dipilih, maka pada setiap *step*, *Q-Value* untuk status dan aksi yang bersangkutan akan diperbarui. Perbaruan tersebut dilakukan berdasarkan *Bellman Equation* yang menggunakan *learning rate* dan *temporal differences* dalam perhitungannya. *Temporal Differences* merepresentasikan seberapa besar perubahan harus dilakukan terhadap *Q-Value* tersebut. Berikut merupakan formula dari *temporal differences*, serta beberapa alternatif *Bellman Equation*.



Temporal Differences



Bellman Equation (1)



Bellman Equation (2)

Setelah nilai *Q-Value* diperbarui, agen kemudian akan melakukan atau mengaplikasikan aksi tersebut di dalam lingkungannya. Dengan demikian, agen telah melakukan transisi ke status yang baru dan bisa jadi menerima *reward* atau *punishment* tertentu berdasarkan aksi tersebut. *Reward* ini diakumulasi oleh agen tersebut hanya untuk *episode* tersebut saja. Jika ternyata pada status yang baru ini agen telah mencapai *terminal state*, maka iterasi pada *episode* tersebut akan dihentikan dan dilanjutkan ke *episode* berikutnya.
Pada setiap *episode* nya, maka nilai *Epsilon* akan diperbarui berdasarkan *decay rate* yang telah ditetapkan sebelumnya. Hal ini dilakukan supaya agen secara bertahap bisa mengurangi kecenderungan untuk melakukan eksplorasi yang mungkin saja menyebabkan algoritma tidak pernah mencapai hasil yang optimal. Nilai epsilon yang baru dapat diimplementasikan seperti berikut.

```
self.epsilon = self.min_epsilon + (
    self.max_epsilon - self.min_epsilon
) * np.exp(-self.decay_rate * episode)
```

Untuk setiap *episode* nya pula, akumulasi *reward* yang didapat bisa dicatat untuk dikalkulasi rataannya nanti. Ketika memasuki *episode* baru, akumulasi *reward* ini akan di-*reset* menjadi 0 lagi. Algoritma inilah yang menjadi algoritma *Q-Learning* dalam *training mode*. Harapannya, pada saat agen telah menyelesaikan *training* nya, maka *policy* yang didapat dalam bentuk *Q-Table* sudah dalam bentuk yang optimal.
Untuk menguji *policy* tersebut, maka agen bisa diuji dalam *inference mode* dengan menggunakan *Q-Table* yang ada. Tanpa menggunakan *Exploration-Exploitation Tradeoff*, maka agen sejak awal akan mencoba untuk melakukan eksploitasi terhadap lingkungannya. Hal ini dilakukan dengan memilih aksi pada status tertentu dengan nilai *Q-Value* tertinggi. Dengan demikian, bisa diamati apabila *Q-Table* pada setiap status dan aksi tertentu yang didapat dari hasil latihan tadi telah optimal atau tidak.

2. Sebutkan kelemahan dari algoritma Q-learning

Salah satu kelemahan yang mungkin dihadapi oleh implementasi *Q-Learning* adalah algoritma yang dijalankan bisa memakan banyak waktu komputasi ketika *rewards* yang diberikan bersifat *sparse* dan ruang dari status yang ada pada lingkungan berukuran sangat besar. Dengan demikian, agen harus berusaha untuk mengunjungi satu persatu setiap status yang ada dan hal ini tentunya akan memakan waktu yang sangat lama.
Selain itu, pada buku Sutton dan Barto tentang *Reinforcement Learning*, salah satu masalah utama yang dimiliki oleh *Q-Learning* adalah *The Deadly Triad*. Hal ini merujuk pada beberapa permasalahan berikut pada *Q-Learning* yang menyebabkan *Q-Learning* menjadi sangat tidak stabil secara teoretis.

a. *Q-Learning* merupakan suatu prosedur yang bersifat *bootstrap*. Artinya, proses pencarian *policy* pada *Q-Learning* merupakan efek domino yang didapat dari hasil tebakan setelah tebakan lainnya. Tebakan ini bersifat *delayed* dan juga *noisy*. Artinya, pada tahap tertentu bisa jadi *policy* yang ada untuk menghasilkan suatu tebakan tertentu direvisi lagi berdasarkan apa yang sedang dihadapi oleh agen pada tahap tersebut dan tentunya hal ini mengakibatkan setiap tebakan yang ada rawan terhadap kesalahan dan subjektivitas.
b. *Q-Learning* merupakan *Off-Policy Algorithm*. Hal ini berarti bahwa *Q-Learning* menggunakan *policy* yang berbeda untuk mengeksplor suatu aksi dari *target policy* yang optimal. Dengan demikian, hal ini membuat *Q-Learning* menjadi kurang *reliable* dan sangat tidak stabil, misalnya ada kasus di mana *Q-Learning* melakukan perbaruan *Q-Value* yang nilainya mencapai tak hingga.

c. *Q-Learning* akan sangat sulit untuk mencari *policy* yang paling optimal ketika lingkungan yang dimilikinya bersifat non-stasioner dan cenderung berubah-ubah. Dengan demikian, *Q-Learning* akan sangat terdampak apabila terdapat fungsi aproksimasi di dalam algoritmanya.