Laporan Teori - Chapter 18: Reinforcement Learning

Pendahuluan

Reinforcement Learning (RL) atau Pembelajaran Penguatan adalah cabang dari Machine Learning yang sangat berbeda dari *supervised* dan *unsupervised learning*. Dalam RL, sebuah sistem cerdas yang disebut agent belajar bagaimana berperilaku dalam sebuah environment (lingkungan) dengan cara melakukan action (aksi) dan menerima reward (hadiah) atau *penalty* (hukuman) sebagai umpan balik. Tujuannya adalah untuk belajar sebuah strategi, yang disebut policy, yang dapat memaksimalkan total *reward* kumulatif dari waktu ke waktu.

RL berfokus pada pengambilan keputusan sekuensial dan seringkali dihadapkan pada credit assignment problem: ketika agent menerima *reward* (terutama yang tertunda), sulit untuk menentukan aksi mana dari rangkaian aksi sebelumnya yang paling berkontribusi terhadap *reward* tersebut.

Terminologi Kunci:

- Agent: Entitas yang belajar dan membuat keputusan.
- Environment: Dunia tempat agent berinteraksi.
- State (s): Representasi dari keadaan lingkungan pada satu waktu.
- Action (a): Keputusan yang diambil oleh agent.
- Reward (r): Sinyal umpan balik numerik yang diterima agent dari lingkungan.
- Policy (π): Strategi atau "otak" dari agent, yang memetakan *state* ke *action*. Tujuan RL adalah menemukan policy optimal, π*.

1. Policy Search dan Policy Gradients (PG)

Salah satu pendekatan dalam RL adalah Policy Search, di mana kita mencari parameter dari sebuah *policy* secara langsung untuk memaksimalkan *reward*. Metode yang paling populer dalam kategori ini adalah Policy Gradients (PG).

Konsep Inti PG:

Dalam PG, policy biasanya direpresentasikan oleh sebuah Jaringan Saraf Tiruan (JST) yang menerima state sebagai input dan menghasilkan probabilitas untuk setiap aksi yang mungkin diambil (stochastic policy).

Proses Pelatihan PG:

1. Agent memainkan "permainan" (disebut *episode* atau *rollout*) beberapa kali menggunakan *policy*-nya saat ini.
2. Setelah setiap *episode*, agent mengevaluasi aksi-aksi yang telah diambil. Aksi yang diikuti oleh *reward* positif dianggap "baik", sedangkan yang diikuti oleh *reward* negatif dianggap "buruk".

3. Algoritma kemudian menggunakan Gradient Descent untuk memperbarui bobot JST. Tujuannya adalah untuk meningkatkan probabilitas dari aksi-aksi "baik" dan menurunkan probabilitas dari aksi-aksi "buruk".

Secara efektif, PG mendorong *policy* untuk lebih sering mengambil tindakan yang terbukti menghasilkan hasil yang positif. Untuk menyeimbangkan antara eksplorasi (mencoba aksi baru) dan eksploitasi (menggunakan aksi yang sudah diketahui baik), PG secara alami menjelajahi lingkungan karena sifatnya yang stokastik.

2. Value-Based Methods: TD Learning dan Q-Learning

Pendekatan lain yang sangat populer adalah *value-based methods*. Alih-alih belajar *policy* secara langsung, metode ini bertujuan untuk mempelajari seberapa "baik" atau berharganya setiap pasangan *state-action*.

Q-Value dan Bellman Equation

Metode ini berpusat pada konsep action-value atau Q-value, yang dinotasikan sebagai Q(s, a). Q-value merepresentasikan total *reward* masa depan yang diharapkan jika agent memulai dari *state* s, mengambil *action* a, dan kemudian mengikuti *policy* optimal sesudahnya.

Hubungan antara Q-value dari satu langkah ke langkah berikutnya dijelaskan oleh Bellman Equation yang fundamental:

$Q^*(s, a) = r + \gamma * \max_{a'} Q^*(s', a')$

Di mana:

- $Q^*(s, a)$ adalah Q-value optimal untuk pasangan (s, a).
- r adalah *reward* yang diterima setelah melakukan aksi a di state s.
- γ (gamma) adalah discount factor ($0 < \gamma < 1$), yang menentukan seberapa penting *reward* masa depan. Nilai yang lebih kecil membuat agent lebih "mementingkan hasil jangka pendek".
- s' adalah *state* berikutnya.
- $\max_{a'} Q^*(s', a')$ adalah Q-value maksimum yang bisa didapatkan dari *state* berikutnya, s'.

Algoritma Q-Learning

Q-Learning adalah algoritma *Temporal Difference (TD) learning* yang menggunakan Bellman Equation untuk secara iteratif memperbarui estimasi Q-value.

Proses Q-Learning:

1. Inisialisasi tabel Q (Q-Table) untuk semua kemungkinan pasangan (state, action) dengan nilai nol.
2. Untuk setiap langkah waktu:

   a. Agent mengamati state s dan memilih action a (biasanya menggunakan strategi ε-greedy untuk menyeimbangkan eksplorasi dan eksploitasi).

b. Agent menerima reward r dan state baru s'.

c. Agent memperbarui nilai Q(s, a) menggunakan rumus pembaruan TD:

Q_baru(s, a) = (1-α) * Q_lama(s, a) + α * (r + γ * max_a' Q(s', a'))

di mana α (alpha) adalah *learning rate* yang mengontrol seberapa besar pembaruan yang dilakukan.

Setelah algoritma konvergen, *policy* optimal dapat dengan mudah diekstraksi dengan memilih aksi yang memiliki Q-value tertinggi untuk setiap *state*.

3. Deep Q-Networks (DQN)

Kelemahan Q-Learning tradisional adalah ia bergantung pada Q-Table, yang hanya praktis untuk lingkungan dengan jumlah *state* dan *action* yang sangat terbatas. Untuk lingkungan yang kompleks dengan ruang *state* yang sangat besar atau kontinu (seperti dari piksel gambar), pendekatan ini tidak mungkin dilakukan.

Deep Q-Network (DQN) adalah solusi dari DeepMind yang menggabungkan Q-Learning dengan Deep Learning. DQN menggunakan Jaringan Saraf Dalam untuk mengaproksimasi fungsi Q-value. Jaringan ini menerima *state* sebagai input dan menghasilkan estimasi Q-value untuk setiap aksi yang mungkin dilakukan di *state* tersebut.

Untuk menstabilkan proses pelatihan yang notabene tidak stabil, DQN memperkenalkan dua teknik kunci:

1. Experience Replay: Alih-alih melatih jaringan pada pengalaman yang berurutan (yang sangat berkorelasi), agent menyimpan setiap pengalaman (s, a, r, s') dalam sebuah memori yang disebut replay buffer. Selama pelatihan, agent mengambil sampel *mini-batch* secara acak dari *buffer* ini. Ini memecah korelasi antar sampel dan membuat pelatihan lebih stabil.
2. Target Network: DQN menggunakan dua jaringan terpisah. Online network adalah jaringan yang terus-menerus dilatih pada setiap langkah. Target network adalah salinan dari *online network* yang bobotnya diperbarui secara berkala (misalnya, setiap beberapa ribu langkah). *Target network* digunakan untuk menghitung nilai target (r + γ * max_a' Q(s', a')) dalam *loss function*. Dengan memiliki target yang lebih stabil dan tidak berubah di setiap iterasi, ini membantu mencegah osilasi dan membuat pelatihan konvergen lebih baik.

Kesimpulan

Reinforcement Learning menawarkan kerangka kerja yang kuat untuk melatih agent agar dapat membuat keputusan optimal dalam lingkungan yang kompleks. Dua keluarga algoritma utama yang mendominasinya adalah Policy Gradients, yang secara langsung belajar sebuah *policy* probabilitas, dan Value-Based Methods seperti Q-Learning dan DQN, yang belajar untuk mengestimasi nilai dari setiap aksi. DQN, dengan kemampuannya menangani ruang *state*

berdimensi tinggi, telah menjadi landasan bagi banyak keberhasilan luar biasa dalam RL, mulai dari bermain game Atari hingga menguasai Go.