1. **Cách xây dựng feature phụ thuộc hành động (action-dependent features) để xấp xỉ giá trị hành động (action values)**
2. **Giải thích công thức cập nhật cho Episodic Sarsa với function approximation**
3. **Giải thích công thức cập nhật cho Expected Sarsa và Q-learning với function approximation**
Hiểu rõ các mục tiêu này là tiền đề để triển khai thuật toán kiểm soát (control) trong Reinforcement Learning (RL) khi không thể lưu bảng Q(s,a) đầy đủ vì không gian trạng thái/hành động lớn hoặc liên tục cs.mcgill.ca stackoverflow.com .

---

2. Từ State-Value đến Action-Value với Function Approximation

Trong prediction (ước lượng value) ta quan tâm đến V(s). Tuy nhiên với control (Sarsa, Q-learning) cần xấp xỉ action-value Q(s,a). Slide nhắc:

- **Giá trị ước lượng** do một hàm tham số $\hat{q}(s, a; w)$ cho trước, với vector trọng số w và feature vector $\phi(s, a)$.
- Với state-value, feature chỉ phụ thuộc state: $\phi(s)$. Nhưng để xấp xỉ Q(s,a), feature phải mang thông tin cả state và action: $\phi(s, a)$.

2.1. Xây dựng feature phụ thuộc hành động (Action-dependent features)

Có hai cách phổ biến:

1. **Stacked features (feature chồng)**:
   - Giả sử ta đã có bộ feature $\phi_{\text{state}}(s)$ (ví dụ coarse coding, tile coding, neural embedding) kích thước d. Nếu có k hành động, ta tạo feature vector $\phi(s, a)$ kích thước $d \times k$ bằng cách "stack" k khối mỗi khối dài d. Với khối thứ i tương ứng hành động i:

$$\phi(s, a) = \begin{cases} [0, 0, \ldots, 0, \ \phi_{\text{state}}(s), \ 0, \ldots, 0]^{\top} & \text{nếu } a \text{ là hành động i} \\ (\text{zero ở các phần khác, chỉ segment i chứa } \phi_{\text{state}}(s)) \end{cases}$$

   - Khi đó, trọng số w cũng có dạng stack k segment $w = [w_1; w_2; \ldots ; w_k]$, mỗi w_i dùng để ước lượng $Q(s, a = i) = w_i^{\top} \phi_{\text{state}}(s)$. Với feature chồng, khi tính Q(s,a), chỉ segment tương ứng action active tham gia dot product, các phần khác không ảnh hưởng.
   - **Ưu điểm**: Cấu trúc rõ ràng, đơn giản triển khai cho linear approximation. Sparse: mỗi $\phi(s, a)$ chỉ active trong segment của action, giúp update chỉ ảnh hưởng segment tương ứng.
   - **Ví dụ**: Slide cho ví dụ: bốn feature state và ba hành động → feature kích thước $4 \times 3 = 12$. Khi ở state s, $\phi_{\text{state}}(s)$ là vector 4 chiều; để ước lượng Q(s, a=0), ta tạo $\phi(s, 0) = [\phi_{\text{state}}(s); 0; 0]$, Q = dot($w$, ϕ). Với a=1, segment offset 4; a=2, offset 8. doutoury2.tistory.com .
   - Khi cần tính Q cho tất cả hành động tại state s, ta trích segment tương ứng từ w và dot product với $\phi_{\text{state}}(s)$.
2. **Neural Network với nhiều output hoặc input (state, action)**:

- **Multi-output network**: Mạng neural đầu vào chỉ state; ở lớp cuối có k neuron output, mỗi output ước lượng Q(s,a=i). Về bản chất, mạng học embedding state rồi riêng biệt dùng cùng embedding để dự đoán giá trị cho từng action.
- **Single-output network đầu vào cả state và action**: Đưa (s,a) vào mạng (ví dụ one-hot action kèm embedding state), output duy nhất là Q(s,a). Cách này thường tốn thời gian nếu số action lớn vì phải gọi forward cho mỗi action khi cần chọn action hay ước lượng max. Tuy nhiên linh hoạt khi action liên tục hoặc muốn mạng đồng nhất xử lý state-action.
- Với tile coding, tương tự có thể xây dựng feature $\phi(s,a)$ bằng cách kết hợp thông tin action (one-hot) và thông tin state (tile-coded), sau đó dot với trọng số hoặc input network. stackoverflow.com .

---

3. Episodic Sarsa với Function Approximation

3.1. Công thức cập nhật (Semi-gradient Episodic Sarsa)

- Sarsa (State-Action-Reward-State-Action) là on-policy TD control. Với function approximation, ta xấp xỉ $\hat{q}(s,a;w)$.
- Trong episodic setting: môi trường có các tập (episodes) kết thúc tại một terminal state. Mỗi episode khởi đầu state đầu, chạy đến terminal.
- **Pseudo-code cơ bản**:
  1. Khởi tạo w (có thể zero hoặc nhỏ ngẫu nhiên).
  2. Với mỗi episode:
     - Khởi tạo state S_0. Chọn action A_0 theo policy ε-greedy dựa trên $\hat{q}$.
     - Với mỗi bước t trong episode:
       - Thực hiện action A_t, nhận reward R_{t+1}, next state S_{t+1}.
       - Chọn A_{t+1} theo policy on-policy (ε-greedy trên $\hat{q}(S_{t+1},\cdot;w)$).
       - Tính TD error:

$$\delta_t = R_{t+1} + \gamma\,\hat{q}(S_{t+1},A_{t+1};w) \;-\; \hat{q}(S_t,A_t;w).$$

       - Cập nhật weights (semi-gradient):

$$w \leftarrow w + \alpha\,\delta_t\,\nabla_w \hat{q}(S_t,A_t;w).$$

       - Nếu S_{t+1} là terminal, kết thúc episode.
- Với linear approximation and stacked features: $\hat{q}(s,a;w) = w^\top \phi(s,a)$, gradient $\nabla_w \hat{q} = \phi(s,a)$. Do đó cập nhật đơn giản:

$$w \leftarrow w + \alpha\,\delta_t\,\phi(s,a).$$

- **Giải thích chi tiết**:

- Cập nhật dựa trên sample TD: reward hiện tại + giá trị ước lượng của next state-action under on-policy. Vì on-policy, action next A_{t+1} được chọn theo cùng policy ε-greedy dựa trên $\hat{q}$.
- Semi-gradient: bỏ phần derivative của target $\hat{q}(S_{t+1}, A_{t+1}; w)$ theo w (chain rule bỏ), nên không phải full gradient descent trên một loss global, nhưng trong linear on-policy đảm bảo hội tụ đến TD fixed point cs.mcgill.ca .
- Policy ε-greedy: khi chọn action next, policy kết hợp khám phá (exploration) và khai thác (exploitation). Parameter ε thường giảm dần theo episode để cuối cùng converge.

- **Convergence**:
  - Với linear on-policy Sarsa (semi-gradient), nếu feature bounded, phân phối trạng thái và action đủ khám phá (policy ε-greedy with sufficient exploration), step-size nhỏ, thường hội tụ đến gần một chính sách ổn định (under assumptions) nhưng không đảm bảo global optimal if function approximation không đủ expressive.
  - Off-policy hoặc phi tuyến có thể không hội tụ ổn định (deadly triad). Đối với off-policy (ví dụ Q-learning), cần kỹ thuật thêm (replay buffer, target network, Gradient-TD).

3.2. Xây dựng feature cho Sarsa

- Với stacked feature: mỗi $\phi(s, a)$ chỉ active phần segment action tương ứng. Khi tính gradient, update chỉ ảnh hưởng đến trọng số segment đó.
- Với neural network multi-output: gradient là backprop qua mạng, cập nhật các tham số mạng hướng giảm TD error.
- Với mạng đầu vào cả (s,a): input kết hợp state và action one-hot; gradient cập nhật toàn bộ tham số mạng. Khi action liên tục, cần mạng đầu ra scalar Q(s,a).

---

4. Expected Sarsa và Q-learning với Function Approximation

Sau khi hiểu Episodic Sarsa, slide mở rộng sang Expected Sarsa và Q-learning.

4.1. Expected Sarsa với Function Approximation

- **Khái niệm**: Thay vì dùng sample action next A_{t+1} như Sarsa, Expected Sarsa dùng **kỳ vọng** giá trị của next state dưới policy hiện tại (target policy).
- **Công thức update**:

$$\delta_t = R_{t+1} + \gamma \sum_{a'} \pi(a' \mid S_{t+1})\, \hat{q}(S_{t+1}, a'; w) \; - \; \hat{q}(S_t, A_t; w),$$

  và

$$w \leftarrow w + \alpha\, \delta_t\, \nabla_w \hat{q}(S_t, A_t; w).$$

- **Ưu/nhược**:
  - **Ưu**: Giảm variance so với Sarsa do tính expectation over actions; ổn định hơn.

- **Nhược**: Tốn tính toán hơn vì cần tính Q(s',a') cho mọi a' và tính weighted sum theo π. Với số action nhỏ, overhead chấp nhận được; với action lớn, có thể expensive.

- **Policy**: Policy on-policy, ví dụ ε-greedy: $\pi(a|s) = \frac{\epsilon}{|A|} + (1-\epsilon)\mathbf{1}\{a = \arg\max_{a'} \hat{q}(s,a')\}$. Khi tính expectation: $\sum_{a'} \pi(a'|s')\hat{q}(s',a')$.
- **Convergence**: Với linear on-policy Expected Sarsa, kỳ vọng giảm variance, hội tụ tốt đến TD fixed point tương ứng.

4.2. Q-learning với Function Approximation

- **Khái niệm**: Off-policy TD control. Update sử dụng max over next-action values, không phụ thuộc policy hành động.
- **Công thức update**:

$$\delta_t = R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a'; w) \; - \; \hat{q}(S_t, A_t; w),$$

$$w \leftarrow w + \alpha\, \delta_t\, \nabla_w \hat{q}(S_t, A_t; w).$$

- **Ưu/nhược**:
  - **Ưu**: Tập trung học tối đa hóa return (theo off-policy), thường hội tụ đến ước lượng Q* nếu bảng (tabular) hoặc với linear under certain conditions.
  - **Nhược**: Off-policy + bootstrap + function approximation có thể dẫn đến divergence (deadly triad) nếu không đầy đủ các điều kiện (ví dụ linear with on-policy distribution? Off-policy là vấn đề). Với linear và certain feature, thỏa điều kiện Tsitsiklis & Van Roy có thể hội tụ, nhưng general phi tuyến (NN) dễ không ổn định.
  - Giải pháp: Deep Q-Network sử dụng replay buffer, target network, clip gradient để ổn định. Với linear small-scale, Q-learning thường ổn định hơn phi tuyến.
- **Policy hành động**: Mặc dù update off-policy (dựa trên max), tuy nhiên agent thường vẫn khám phá với ε-greedy behavior policy để thu thập dữ liệu. Chính sách target là greedy.
- **Implementation**: Với stacked feature hoặc neural multi-output, update tương tự Sarsa nhưng dùng max. Khi tính max, cần trích Q(s', a') cho mọi a'.

---

5. Ví dụ minh họa Sarsa trong Grid World

Slide đưa ví dụ về dùng Sarsa để học chính sách tối ưu cho môi trường grid world:

- **Môi trường**: Grid world đơn giản, trạng thái là ô tọa độ; hành động: đi lên/xuống/trái/phải. Reward: ví dụ -1 mỗi bước, 0 tại terminal hoặc +1 khi đạt goal.
- **Feature representation**:
  - **Tabular**: nếu môi trường nhỏ, có thể dùng tabular Q(s,a).
  - **Function approximation**: giả sử thêm dimension (liên tục) hoặc muốn generalize, dùng tile coding trên tọa độ state, kết hợp action bằng stacked features.
- **Thuật toán**:

1. Khởi tạo w.
2. Với mỗi episode: khởi state, chọn action ε-greedy, vòng lặp Sarsa update như mục 3.1.
3. Policy dần cải thiện khi w cập nhật.

- **Quan sát**:
  - Nếu dùng feature coarse/tile coding, Sarsa học nhanh hơn tabular khi không gian lớn hoặc muốn generalize sang state chưa trải nghiệm.
  - Nếu feature kém, policy ước lượng sai, dẫn đến learning kém.
- **Expected Sarsa**: nếu tính expectation, learning smooth hơn, ít nhiễu hơn do không phụ thuộc vào action sample next.
- **Q-learning**: có thể học Q* off-policy, tuy nhiên với function approximation linear, cần chú ý convergence.

---

6. Cân nhắc khi dùng Function Approximation

6.1. Linear vs Phi tuyến

- **Linear**: $\hat{q}(s,a) = w^\top \phi(s,a)$. Với feature phù hợp, linear đảm bảo hội tụ trong on-policy Sarsa/Expected Sarsa; off-policy Q-learning có thể hội tụ nếu phân phối trải nghiệm phù hợp và feature thỏa điều kiện (Tsitsiklis & Van Roy). Linear dễ debug, nhanh, interpretability cao.
- **Phi tuyến (Neural Network)**: $\hat{q}(s,a)$ do mạng nơ-ron. Có khả năng biểu diễn phức tạp hơn, phù hợp với trạng thái hình ảnh hoặc high-dimensional. Tuy nhiên dễ gặp instability (deadly triad). Cần kỹ thuật như replay buffer, target network, gradient clipping, điều chỉnh learning rate, sử dụng Double Q-learning, Dueling architecture để giảm bias và tăng ổn định.
- **Semi-gradient**: Cả linear và phi tuyến, cập nhật vẫn semi-gradient: bỏ derivative của bootstrap target. Với phi tuyến, điều này đôi khi gây khó hội tụ; các phương pháp Gradient-TD (GTD) hoặc True Online TD có thể cải thiện stability trong một số tình huống prediction off-policy, nhưng cho control complexity tăng.

6.2. Feature phụ thuộc hành động với tile coding hoặc coarse coding

- Với discrete action nhỏ, stacked tile coding thường hiệu quả.
- Với action liên tục, cần mạng hoặc khác (actor-critic, policy gradient).

6.3. Policy và Exploration

- Với Sarsa/Expected Sarsa (on-policy), policy ε-greedy phải đảm bảo "sufficient exploration" (thăm hết state-action) để weights hội tụ đến ước lượng tốt.
- Với Q-learning (off-policy), behavior policy vẫn phải khám phá (ε-greedy) để thu đủ dữ liệu, dù update dùng max.
- Annealing ε: Thường giảm ε dần theo episode để cuối cùng converge policy ổn định.

6.4. Hyperparameters

- **Learning rate (α)**: Với linear, α nhỏ để đảm bảo hội tụ, nhưng không quá nhỏ để học chậm. Với phi tuyến, tỉ mỉ điều chỉnh, có thể dùng adaptive optimizers (Adam), nhưng vẫn cần careful tuning.
- **Discount factor (γ)**: Chọn phù hợp bài toán.
- **Số episode, length episode**: Với episodic, quan sát learning curves (return theo episode) để điều chỉnh hyperparameters.
- **Regularization**: Với linear, có thể thêm weight decay; với NN, dropout, L2 regularization nếu cần tránh overfitting dữ liệu tương tác (thường noise do environment).

6.5. Convergence và stability

- **Linear on-policy Sarsa/Expected Sarsa**: có lý thuyết hội tụ đến TD fixed point nếu phân phối trải nghiệm đủ "ergodic" và step-size giảm dần hoặc small constant.
- **Off-policy Q-learning + Linear**: có điều kiện hội tụ nếu policy behavior có full support và feature cho phép projected Bellman operator contraction; Tsitsiklis & Van Roy (1997) phân tích.
- **Phi tuyến**: không đảm bảo hội tụ; DQN và các biến thể giảm thiểu instability qua các trick thực tiễn.
- **Deadly triad**: Off-policy + bootstrap + function approximation có thể divergence. Cần awareness khi implement.

---

7. Ví dụ code minh họa (pseudocode)

Dưới đây là pseudocode tóm tắt Episodic Sarsa với stacked linear features:

python Sao chép Chỉnh sửa

```
# Giả sử: # - phi_state(s): trả về feature vector state dimension d # - actions = [0,1,...,k-1] # - stacked feature: phi(s,a) xây dựng stack length d*k, active segment tương ứng # Khởi weights w = np.zeros(d * k) alpha = 0.01 gamma = 0.9 epsilon = 0.1 def get_phi_sa(s, a): phi = np.zeros(d * k) idx = a * d phi[idx: idx + d] = phi_state(s) return phi def q_value(s, a): return np.dot(w, get_phi_sa(s, a)) def choose_action(s): if np.random.rand() < epsilon: return np.random.choice(actions) # greedy q_vals = [q_value(s,a) for a in actions] return int(np.argmax(q_vals)) for episode in range(num_episodes): s = env.reset() a = choose_action(s) done = False while not done: s_next, r, done, info = env.step(a) if done: # terminal: target = r (no next Q) target = r delta = target - q_value(s, a) grad = get_phi_sa(s, a) w += alpha * delta * grad break # chọn a_next on-policy a_next = choose_action(s_next) target = r + gamma * q_value(s_next, a_next) delta = target - q_value(s, a) grad = get_phi_sa(s, a) w += alpha * delta * grad s, a = s_next, a_next # Có thể giảm epsilon theo episode nếu muốn
```

- Với Expected Sarsa, thay dòng target = r + gamma * q_value(s_next, a_next) thành:

```python
# compute expectation under policy π (ε-greedy) q_vals_next = np.array([q_value(s_next,a2) for a2 in actions]) # probability under ε-greedy: π(a*|s') = 1-ε + ε/|A| if a* is argmax, else ε/|A| # Tính expectation: probs = np.ones(k) * (epsilon / k) best = np.argmax(q_vals_next) probs[best] += 1 - epsilon exp_q = np.dot(probs, q_vals_next) target = r + gamma * exp_q
```

- Với Q-learning, thay target = r + gamma * max_a' q_value(s_next,a').

Nếu dùng neural network multi-output, pseudocode tương tự nhưng gradient tính qua backprop thay vì dot product với phi.

---

8. Đánh giá, Theo dõi và Debug

- **Monitoring**:
  - Theo dõi return trung bình mỗi episode để xem policy cải thiện.
  - Theo dõi giá trị Q ước lượng tại một số state cố định để xem hội tụ behavior.
  - Theo dõi TD error trung bình để đánh giá stability.
- **Debug tips**:
  - Kiểm tra feature φ(s,a) có đúng kích thước và active đúng segment.
  - Khởi weights phù hợp, khởi alpha nhỏ.
  - Kiểm tra chọn action ε-greedy: đảm bảo exploration đầy đủ, tránh “chết” ở vùng nào đó.
  - Với tile coding, kiểm tra mapping từ state sang tile indices có logic.
  - Quan sát learning curves: nếu diverge, giảm learning rate hoặc kiểm tra gradient/numerical instability.