Preguntas y respuestas para defensa oral

Proyecto: DQN aplicado a CarRacing-v3 (Gymnasium). Fecha: 2025-12-20. Formato: A4.

Este documento reúne preguntas típicas que una profesora podría hacer sobre el proyecto y respuestas sugeridas basadas en la implementación del repositorio.

1. ¿Cuál es el objetivo del proyecto y por qué CarRacing-v3 es un buen caso de estudio?

El objetivo es entrenar un agente de Aprendizaje por Refuerzo (RL) que aprenda a conducir en CarRacing-v3 usando observaciones visuales. CarRacing-v3 es un entorno desafiante porque la observación es una imagen (96×96) y el control original es continuo (dirección, aceleración, freno), lo que obliga a tomar decisiones de diseño (preprocesamiento, discretización de acciones, estabilidad del entrenamiento).

2. ¿Qué algoritmo de RL se implementó y cuál es la idea central?

Se implementó DQN (Deep Q-Network). La idea central es aproximar con una red neuronal la función de valor-acción Q(s,a) y aprenderla minimizando el error temporal-diferencial: la red ajusta sus Q-valores para que se acerquen a un objetivo de Bellman r + γ max_a' Q(s',a').

3. ¿Por qué DQN requiere acciones discretas y qué hicieron con las acciones continuas de CarRacing?

DQN en su forma clásica está formulado para un conjunto discreto de acciones porque produce un vector de Q-valores, uno por acción. CarRacing tiene control continuo, así que se discretizó el espacio de acciones en 12 acciones fijas (combinaciones de steer/gas/brake) para poder aplicar DQN.

4. ¿Qué limitaciones introduce la discretización de acciones?

Reduce expresividad: el agente solo puede elegir entre 12 comandos predefinidos, lo que puede limitar la suavidad de la conducción. También puede introducir sesgo: si la discretización no cubre bien maniobras críticas, el agente no podrá aprenderlas aunque exista señal de recompensa.

5. ¿Cómo se representa el estado del entorno para la red?

El estado se construye a partir de la observación visual preprocesada: se convierte a escala de grises, se normaliza a [0,1] y se apilan 4 frames consecutivos. Así el tensor final tiene forma (4, 96, 96) y captura información temporal de movimiento.

6. ¿Por qué se usa frame stacking (apilar frames) en lugar de un solo frame?

Un solo frame no contiene velocidad ni dirección de movimiento de forma explícita. Al apilar varios frames consecutivos, la red puede inferir dinámica temporal (por ejemplo, si el auto se aproxima a un borde o si está girando), lo que suele mejorar el desempeño en entornos visuales.

7. ¿Qué arquitectura de red usa el agente?

Una CNN sencilla: dos convoluciones con ReLU y MaxPool, seguida de capas densas. La red recibe (4,96,96) y devuelve 12 Q-valores (uno por acción discretizada).

8. ¿Qué es el replay buffer y para qué sirve?

El replay buffer almacena transiciones (s, a, r, s', done). En lugar de aprender solo con la transición más reciente, se muestrean minibatches aleatorios del buffer, reduciendo correlación temporal y estabilizando el entrenamiento. En el proyecto el buffer tiene capacidad 50,000.

9. ¿Cómo se define la transición almacenada en memoria?

Se guarda (state, action_index, reward, next_state, done). En el código, la acción se guarda como índice (no como tupla completa) para hacer la transición más compacta y fácil de muestrear.

10. ¿Qué es la red objetivo (target network) y cómo se actualiza?

La red objetivo es una copia congelada de la red principal, usada para calcular el término max Q(s',a') del objetivo. Se actualiza cada cierto número de episodios copiando los pesos de la red principal. En el proyecto se actualiza cada 5 episodios.

11. ¿Qué política de exploración se usa y cómo evoluciona?

Se usa ε-greedy: con probabilidad ε se elige una acción aleatoria, y con 1-ε la acción con mayor Q. ε empieza en 1.0, baja multiplicándose por 0.995 hasta un mínimo 0.05. En este proyecto ε decae con cada actualización de entrenamiento (replay), por lo que puede bajar rápido si hay muchas actualizaciones.

12. ¿Qué hiperparámetros principales se usan y por qué?

γ=0.99 para valorar recompensas futuras en conducción; lr=1e-4 con Adam, típico para CNNs con entradas visuales; batch_size=64; buffer=50,000; stack=4; frame-skip=2 (repite acciones 3 frames) para reducir cómputo y suavizar control.

13. ¿Qué es frame skipping y cómo está implementado en el entrenamiento?

Frame skipping significa repetir la misma acción durante varios frames para ahorrar cómputo y reducir ruido. En el código se usa SKIP_FRAMES=2, y se ejecuta el paso del entorno en un bucle de SKIP_FRAMES+1, acumulando recompensas.

14. ¿Qué función de pérdida se usa para entrenar la red y qué se está minimizando?

Se usa MSELoss (error cuadrático medio) entre Q_pred (Q(s,a) de la red) y Q_target (r + γ max Q(s',a') de la red objetivo, con corte si done). Se minimiza el error temporal-diferencial.

15. ¿Cómo se calcula el objetivo de Bellman en tu implementación?

Para cada transición: Q_target = r + γ * max_a' Q_target_net(s',a') * (1 - done). Si done=1, el término futuro se anula.

16. ¿Qué métricas se registran durante el entrenamiento y dónde quedan guardadas?

Se guarda un CSV por episodio con: episodio, reward_total, epsilon, tamaño del buffer y loss promedio del episodio. Se guarda en resultados//metricas_entrenamiento.csv. Además se generan gráficos (reward, epsilon, buffer, loss).

17. ¿Cómo es la metodología experimental de tu proyecto (pipeline completo)?

1) Entrenar con main.py y src/entrenamiento.py guardando métricas y checkpoints. 2) Evaluar modelos guardados con tests/probar_modelo.py con ε=0 para ver comportamiento determinista. 3) Analizar resultados y comparar en el cuaderno notebooks/entrenamiento_y_resultados.ipynb usando los CSV y gráficos.

18. ¿Cómo garantizan que la evaluación sea 'justa' y comparable?

En evaluación se fija ε=0 (sin exploración) para usar siempre la política aprendida. Se evalúan episodios con el mismo procedimiento y se guardan métricas por episodio. Para comparaciones más robustas, lo ideal sería repetir con varias semillas y más episodios.

19. ¿Qué hace exactamente tests/probar_modelo.py y qué artefactos produce?

Carga un modelo .pth, ejecuta N episodios con ε=0, registra CSV (episodio, reward_total, frames), genera un gráfico de rewards y opcionalmente crea un GIF del mejor episodio. Crea una carpeta resultados/_/ con logs y salidas.

20. ¿Qué es Gymnasium y qué papel juega en el proyecto?

Gymnasium es una librería estándar para entornos de RL. Provee la API env.reset() y env.step() y define espacios de observación/acción. En este proyecto Gymnasium entrega CarRacing-v3, la dinámica del entorno y la función de recompensa.

21. ¿Cuál es la interfaz step() en Gymnasium y por qué es importante?

Gymnasium retorna (obs, reward, terminated, truncated, info). Es importante separar terminated (fin natural del episodio) de truncated (corte por límite de tiempo u otra condición), y usar done = terminated or truncated para controlar el bucle.

22. ¿Qué decisiones del proyecto están en src/configuracion.py y por qué conviene centralizarlas?

Ahí están hiperparámetros (γ, lr, ε, batch size, SKIP_FRAMES, frecuencias de guardado/target) y parámetros de estado (stack, tamaño). Centralizarlos facilita reproducibilidad, cambios controlados y evita 'valores mágicos' repartidos.

23. ¿Qué rol cumple src/entorno.py?

Encapsula la creación del entorno CarRacing-v3 con el render_mode correspondiente (human si se desea ver la ventana, rgb_array si se entrena rápido o se generan frames). Esto evita duplicar lógica y simplifica main/evaluación.

24. ¿Qué rol cumple src/preprocesamiento.py?

Convierte frames del entorno en una representación adecuada para la CNN: grises, resize a 96×96, normalización, y manejo del stack temporal con deque. También expone funciones para generar el estado final (np.stack).

25. ¿Qué rol cumple src/agente.py?

Contiene la red DQN (PyTorch) y la lógica del agente: selección ε-greedy, memoria (replay buffer), entrenamiento por minibatch (replay), cálculo de objetivos de Bellman y actualización de red objetivo.

26. ¿Qué rol cumple src/entrenamiento.py?

Implementa el loop de entrenamiento por episodios: reset del entorno, construcción de estado, selección de acción, frame-skip, almacenamiento de transición, replay, registro de métricas, guardado de modelos y actualización del target.

27. ¿Por qué el loss en RL puede verse 'inestable' aunque el reward mejore?

Porque la distribución de datos cambia (experiencias del buffer cambian), el objetivo depende de una red (bootstrapping), y la política va cambiando. En RL, el loss no siempre correlaciona con desempeño; por eso se evalúa también con reward y pruebas sin exploración.

28. ¿Qué es MDP (Proceso de Decisión de Markov) y cómo se aplica aquí?

Un MDP define estados, acciones, transiciones y recompensas. Aquí, el 'estado' es una aproximación (stack de frames), las acciones son 12 discretas, la transición la define el simulador físico, y la recompensa la define el entorno por progreso/penalizaciones.

29. ¿Qué significa 'on-policy' vs 'off-policy' y dónde cae DQN?

On-policy aprende sobre la política que ejecuta; off-policy aprende de datos generados por otra política. DQN es off-policy porque aprende desde el replay buffer con transiciones de políticas ε-greedy históricas.

30. ¿Qué mejoras técnicas propondrías para estabilizar/elevar desempeño?

(1) Double DQN para reducir sobreestimación. (2) Dueling DQN. (3) Prioritized Experience Replay. (4) Huber loss y gradient clipping. (5) Actualización suave del target (Polyak). (6) Mejor calendario de ε. (7) Más episodios/semillas para evaluación.

31. Si la profesora pregunta por alternativas a DQN en acciones continuas, ¿qué responderías?

Que para control continuo es común usar algoritmos Actor-Critic como DDPG, TD3 o SAC (continuos), o PPO con política continua. En este proyecto se discretizó para usar DQN, pero una extensión natural sería pasar a SAC/TD3 para evitar la discretización.

32. ¿Qué riesgos de reproducibilidad existen y cómo mitigarlos?

Sin fijar semillas, los resultados pueden variar. Para mitigarlo: fijar seed (NumPy/PyTorch/Gymnasium), registrar versiones de dependencias, guardar hiperparámetros en logs/metadata, y evaluar con varias semillas y promedios.

33. ¿Qué 'casos similares' existen en la literatura o en práctica?

DQN se popularizó en tareas visuales tipo Atari. CarRacing es un benchmark visual similar (aunque con control continuo). En práctica, se suelen aplicar técnicas del pipeline de Atari (preprocesamiento/stacking/replay/target) y luego extender con mejoras como Double DQN, PER y enfoques Actor-Critic.

34. ¿Cómo explicarías, en una frase, la diferencia entre 'aprender' y 'evaluar' en tu repo?

Aprender es actualizar pesos con replay usando ε-greedy (exploración), mientras que evaluar es congelar el modelo y correr episodios con ε=0 para medir desempeño sin aleatoriedad.

35. ¿Cuál es el punto de entrada del proyecto y qué hace?

El punto de entrada es main.py. Parsear argumentos CLI, configurar el logging (archivo en logs/ con timestamp), ajustar render (human/rgb_array) y lanzar el entrenamiento llamando a entrenar() en src/entrenamiento.py.

36. ¿Qué argumentos de línea de comandos soporta main.py y para qué sirven?

Soporta --start/--end (rango de episodios), --epsilon (ε inicial), --gamma, --lr, --batch-size y --nombre-exp para nombrar el experimento (carpetas/logs). También --render on/off para activar ventana. El argumento --model existe como placeholder pero en el estado actual no se usa para reanudar automáticamente.

37. Si la profesora pregunta: '¿por qué el argumento --model no se usa?', ¿qué dirías?

Que se dejó el parámetro pensando en una extensión para reanudar entrenamiento desde un checkpoint, pero aún no se conectó a la función entrenar(). Como mejora, se podría cargar pesos y también restaurar epsilon/optimizador si se guarda ese estado.

38. ¿Cómo se crea la carpeta de resultados durante entrenamiento?

En src/entrenamiento.py se crea resultados// y resultados//modelos/. Allí se guarda metricas_entrenamiento.csv y los modelos .pth periódicamente.

39. ¿Cada cuánto se guardan modelos y cómo se nombran?

Se guardan cada SAVE_TRAINING_FREQUENCY (25) episodios. El nombre es modelo_ep_.pth dentro de resultados//modelos/.

40. ¿Cómo se registran las métricas de entrenamiento y qué columnas tienen?

Se escribe un CSV en resultados//metricas_entrenamiento.csv con columnas: episodio, reward_total, epsilon, buffer, loss. loss es el promedio de losses del episodio (0 si no entrenó ese episodio).

41. ¿Qué significa que el loss promedio sea 0 en algunos episodios?

Que no se ejecutó replay (entrenamiento) en ese episodio, típicamente porque el buffer aún no tiene suficientes transiciones para armar un minibatch (len(memoria) <= batch_size).

42. ¿Por qué en entrenamiento se usa un contador de castigos y un corte por recompensa_total<0?

Es una heurística para terminar episodios que van claramente mal y ahorrar tiempo de cómputo: si tras cierto tiempo la recompensa por step es negativa repetidamente, o si el acumulado cae por debajo de 0, se corta el episodio. Esto acelera pero puede sesgar el dataset del replay.

43. ¿Qué impacto puede tener esa heurística de corte temprano sobre el aprendizaje?

Puede sesgar el buffer hacia trayectorias 'menos malas' y reducir la exposición a estados de recuperación; a veces ayuda a estabilidad y rapidez, pero también puede impedir aprender a salir de situaciones difíciles. Una mejora sería justificarla con experimentos A/B o ajustar umbrales.

44. En src/preprocesamiento.py, ¿qué funciones existen y cuál es su responsabilidad?

process_state_image(frame) convierte a grises, resize y normaliza.
generar_stack_inicial(frame) crea un deque con 4 copias del primer frame procesado.
actualizar_stack(stack, frame) agrega el nuevo frame procesado.
generar_state_frame_stack_from_queue(queue) apila el deque en un numpy array (4,96,96).

45. ¿Cómo aseguras que el stack temporal siempre tenga tamaño 4?

Se usa deque con maxlen=STATE_STACK. Al hacer append, automáticamente se descarta el frame más viejo manteniendo tamaño constante.

46. ¿Qué bug potencial hay en la conversión de color en preprocesamiento?

Gymnasium entrega frames RGB, pero OpenCV por defecto interpreta BGR. El código usa cv2.COLOR_BGR2GRAY; estrictamente, para RGB lo correcto sería cv2.COLOR_RGB2GRAY. En la práctica, la diferencia puede ser pequeña, pero es una mejora importante para dejarlo correcto.

47. En src/agente.py, ¿qué hace seleccionar_accion() exactamente?

Implementa ε-greedy: con probabilidad ε elige una acción aleatoria del action_space; si no, convierte el estado a tensor (agregando batch), evalúa la red q_red, toma argmax y devuelve la acción (tupla) correspondiente.

48. ¿Dónde se convierte la acción elegida a un array que Gymnasium pueda ejecutar?

En src/entrenamiento.py (y en tests/probar_modelo.py) la acción tupla se convierte con np.array(accion_tupla, dtype=np.float32) antes de llamar env.step().

49. ¿Cómo se calcula q_pred en replay y por qué se usa gather?

La red produce pred con shape (batch, num_acciones). gather(1, acciones.unsqueeze(1)) selecciona, para cada fila del batch, el Q correspondiente a la acción realmente ejecutada. Eso produce q_pred para comparar con el objetivo.

50. ¿Qué significa 'done' en tu memoria y cómo lo manejas?

done indica final de episodio. En el target se usa (1 - done) para anular el término futuro si el episodio terminó. En entrenamiento, done se toma de terminado (terminated o truncated, más cortes heurísticos).

51. ¿La red está en GPU o CPU? ¿Cómo lo controla el código?

Por defecto está en CPU porque no se define device ni se llama .to(device). Si se quisiera GPU, habría que definir torch.device('cuda' si disponible) y mover modelos y tensores.

52. ¿Cómo manejas el modo render durante entrenamiento/evaluación?

src/configuracion.py mantiene una variable global RENDER. main.py llama set_render_mode(). src/entorno.py lee RENDER y crea el entorno con render_mode='human' o 'rgb_array'.

53. ¿Qué archivos generan resultados para el informe y cómo se conectan con el notebook?

Durante entrenamiento se generan CSV y gráficos en resultados//. Durante evaluación se generan CSV/grafico_rewards.png/mejor_episodio.gif en resultados/_/. El notebook lee los CSV (por ejemplo eval_200/metricas.csv y eval_1000/metricas.csv) y muestra tablas/gráficos.

54. ¿Por qué en evaluación guardan un GIF del 'mejor episodio' y no de todos?

Para ahorrar tamaño/tiempo. Guardar todos los episodios en GIF puede ser pesado; el 'mejor' sirve como evidencia visual del comportamiento cuando el agente tuvo su mejor

desempeño en esa corrida.

55. ¿Cómo se decide cuál fue el mejor episodio en tests/probar_modelo.py?

Se compara reward_total de cada episodio y se guarda la secuencia de frames del episodio con mayor reward_total.

56. ¿Qué hace el logging y por qué es útil para defensa del proyecto?

Guarda en logs/ y en resultados/ información de hiperparámetros, progreso y métricas. Es útil para reproducibilidad, auditoría del experimento y para responder preguntas sobre 'qué se corrió exactamente'.

57. Si tuvieras que explicar el flujo de datos en entrenamiento (de obs a aprendizaje), ¿cómo lo describes?

obs (RGB) → process_state_image (gris+norm) → stack (4 frames) → estado (4,96,96) → seleccionar_accion → env.step (posible frame-skip) → reward acumulado → actualizar_stack → next_state → memorize → replay (minibatch) → backprop y update de la red.

58. ¿Qué diferencias hay entre 'terminated' y 'truncated' en Gymnasium y cómo afecta a tus episodios?

terminated es fin por condición del entorno (p. ej. fallo); truncated es fin por límite/tiempo. En ambos casos se considera done=True para cortar el loop y para construir targets sin término futuro.

59. ¿Qué mejoras de ingeniería de software propondrías al código?

(1) Separar configuración/CLI en un módulo. (2) Añadir seed y registrar versiones. (3) Añadir soporte real de reanudar entrenamiento (--model). (4) Añadir device CPU/GPU. (5) Tests unitarios para preprocesamiento/shape.