PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
Facultad de Ingeniería
IIC2613 Inteligencia Artificial
2024-2

Tarea 5

Aprendizaje Reforzado y Aprendizaje Profundo

Fecha de entrega: Miércoles 27 de noviembre a las 23:59 hrs

Aspectos generales

Formato y plazo de entrega

El formato de entrega son archivos con extensión `.ipynb` con un PDF para las respuestas teóricas y archivos de extensión `.py` para el apartado de aprendizaje reforzado. El lugar de entrega es en el repositorio de la tarea, en la branch por defecto, hasta el miércoles 27 de noviembre a las 23:59 hrs. Para crear tu repositorio, debes entrar en el enlace del anuncio de la tarea en Canvas. Por último, recuerda que los cupones de atraso son días **no hábiles** extra.

Integridad Académica

Este curso se adhiere al Código de Honor establecido por la universidad, el cual tienes el deber de conocer como estudiante. Todo el trabajo hecho en esta tarea debe ser **totalmente individual**. La idea es que te des el tiempo de aprender estos conceptos fundamentales, tanto para el curso, como para tu formación profesional. Las dudas se deben hacer exclusivamente al cuerpo docente a través de las *issues* en GitHub.

Por otra parte, sabemos que estás utilizando material hecho por otras personas, por lo que es importante reconocerlo de la forma apropiada. Todo lo que obtengas de internet debes citarlo de forma correcta (ya sea en APA, ICONTEC o IEEE). Cualquier falta a la ética y/o a la integridad académica será sancionada con la reprobación del curso y los antecedentes serán entregados a la Dirección de Pregrado.

Comentarios adicionales

El objetivo de esta tarea es que puedan utilizar redes neuronales y redes neuronales convolucionales para llevar a cabo tareas de clasificación sobre conjuntos de datos y el uso del algoritmo Q-Learning para desarrollar una política de comportamiento de un agente. Es fundamental que pongan énfasis en las justificaciones de sus respuestas, cuidando la redacción, ortografía; manteniendo el código ordenado y comentado. **Aquellas respuestas que solo presenten resultados o código** (sin contexto ni comentarios) **no serán consideradas**, mientras que tareas desordenadas pueden ser objeto de descuentos.

1. DCChampiñones Parte II: Clasificador de imágenes de hongos (3 puntos)

IMPORTANTE: Para esta parte está EXPRESAMENTE PROHIBIDO subir el set de datos utilizado a tu repositorio en GitHub. En caso contrario, se aplicará un descuento en tu entrega. Procura subir solamente el notebook y las respuestas de desarrollo.

Introducción

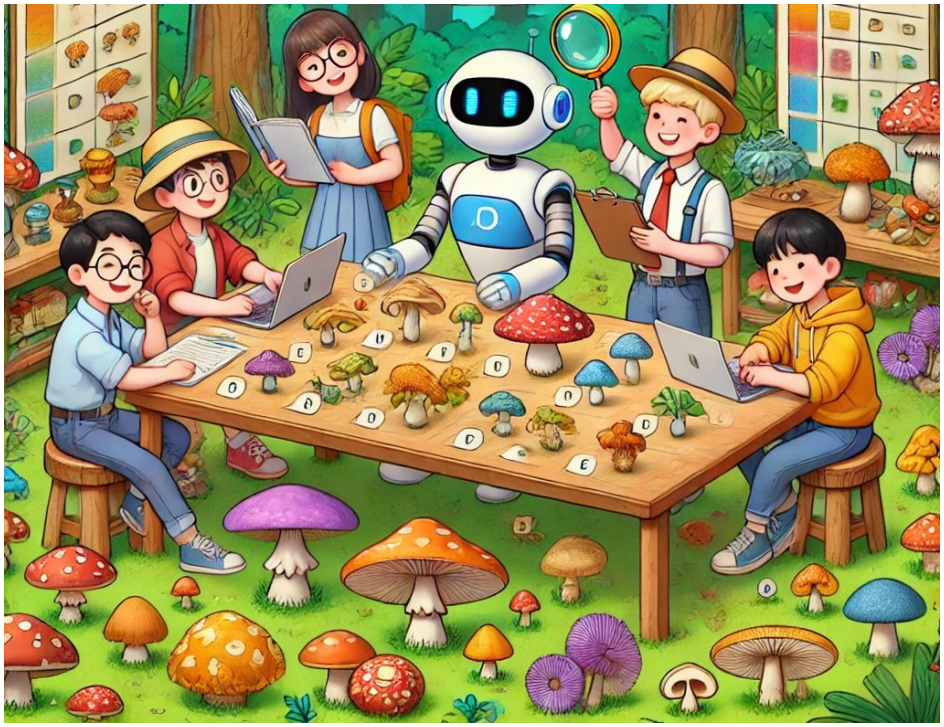

Figura 1: Alumnos trabajando en la clasificación de los hongos

Después de trabajar arduamente en la primera parte del proyecto "DCChampiñones", un grupo de alumnos del curso "Inteligencia Artificial" logró implementar modelos de machine learning para determinar si un hongo era venenoso o no, utilizando datos tabulares. Sin embargo, su aventura no termina ahí. Con el éxito inicial de su emprendimiento y un nuevo objetivo en mente, los estudiantes decidieron llevar su análisis al siguiente nivel.

Ahora, con un mayor entendimiento de las técnicas de deep learning, buscan clasificar los hongos no solo por su peligrosidad, sino también por su tipo específico, utilizando imágenes de los hongos que recolectaron. Recordaron que para abordar problemas de clasificación de imágenes, es posible utilizar redes neuronales de diversos tipos, como las MLP o las redes convolucionales (CNNs), o también pueden aprovechar embeddings generados por modelos pre entrenados, como `ResNet50` para obtener representaciones de las imágenes y así mejorar la clasificación.

Descripción del Problema

El objetivo de esta sección es clasificar imágenes de hongos en sus respectivas clases. Estas imágenes están contenidas en el dataset **Mushrooms** que fue obtenido de **Kaggle**[1] y contiene aproximadamente 7000

[1] https://www.kaggle.com/datasets/lizhecheng/mushroom-classification

imágenes. Este lo podrán encontrar en **Canvas**[2] (Archivos → Tareas → Tarea 5 → Mushroom.zip). A continuación unas imágenes de ejemplo del dataset:

Figura 2: Imagen de un hongo de la clase Hygrocybe

Figura 3: Imagen de un hongo de la clase Amanita

Figura 4: Imagen de un hongo de la clase Boletus

Para la clasificación, tu misión es probar diferentes estrategias para ver cual es más efectiva para esta tarea. En primera instancia debes probar un MLP y una red CNN que reciban como input las imágenes. En segundo lugar, deberás usar las características (embeddings) extraídas mediante un modelo pre entrenado de `ResNet50`, para luego entrenar un nuevo MLP y un SVM que utilice estos embeddings para clasificar las imágenes. Se te proporcionará un cuaderno `.ipynb` con la carga de datos y preprocesamiento de las imágenes, junto con la extracción de lo embeddings ya implementada. Tu objetivo será construir y evaluar los modelos previamente descritos, comparando su rendimiento y analizando los resultados.
A modo de resumen, deberás implementar los siguientes clasificadores:

1. **Clasificador MLP (Imágenes)**: Implementa un Perceptrón Multicapa (MLP) que tome las imágenes preprocesadas (escaladas y aplanadas) como entrada. Ajusta los hiperparámetros para optimizar el rendimiento y evalúa el modelo en el conjunto de prueba.

2. **Clasificador CNN (Imágenes)**: Implementa una Red Neuronal Convolucional (CNN) utilizando `Keras`. Diseña una arquitectura adecuada para el problema de clasificación de imágenes y experimenta con diferentes configuraciones. Evalúa el rendimiento del modelo y presenta las métricas correspondientes.

3. **Clasificador SVM con los Embeddings de ResNet)**: Utiliza la clase `SVC` de `scikit-learn` para construir un clasificador SVM que utilice los embeddings generados con `ResNet50`. Ajusta los parámetros del modelo y evalúa su rendimiento en el conjunto de prueba.

4. **Clasificador MLP con los Embeddings de ResNet)**: Implementa otro Perceptrón Multicapa (MLP), pero que ahora utilice los embeddings como entrada en vez de las imágenes directamente. Ajusta los hiperparámetros y evalúa el modelo en el conjunto de prueba, comparando su rendimiento con el del SVM.

Por otro lado, para evaluar para evaluar los modelos implementados y comparar los resultados usaremos las siguientes métricas ya utilizadas en la tarea 4:

Métricas

Para evaluar el rendimiento, es fundamental utilizar métricas que representen lo que estamos buscando, a continuación presentamos las 4 métricas que utilizaremos en esta sección:

[2] https://cursos.canvas.uc.cl/courses/75222/files/folder/Tareas/Tarea%205?preview=11413352

- **Precision**: La precisión se define como el número de verdaderos positivos dividido por la suma de verdaderos positivos y falsos positivos.
- **Recall**: La sensibilidad o recall mide cuán bien el modelo puede identificar correctamente las instancias positivas. Se define como el número de verdaderos positivos dividido por la suma de verdaderos positivos y falsos negativos.
- **F1-score**: La F1-score es la media armónica entre la precisión y el recall. Proporciona un equilibrio entre estas dos métricas, especialmente útil cuando se busca un compromiso entre precisión y recall.
- **Support**: El soporte (support) indica el número de ocurrencias reales de cada clase en el conjunto de datos. Esta métrica es útil para entender el peso o la relevancia de cada clase en la evaluación global del modelo.

Para el resto de la tarea, llamaremos **reporte de clasificación** a evaluar a los modelos en estas métricas.

Actividades a realizar:

1.1. Análisis del preprocesamiento (0.4 puntos)

Antes de armar una red neuronal, por lo general existe un proceso meticuloso de preparación y comprensión de los datos. Esta fase preliminar, conocida como preprocesamiento, juega un papel crucial en la determinación del éxito de nuestros modelos.

- Investiga y describe por qué el preprocesamiento es una etapa crucial para las redes neuronales. Explica cómo los pasos de preprocesamiento pueden afectar el rendimiento de un modelo.
- Muestra un gráfico de la distribución de imágenes y muestra una imagen por clase. Luego revisa imágenes de diferentes clases. Como podrás darte cuenta, existe una variabilidad visual entre las imágenes que pertenecen a la misma clase, es decir, dentro de una clase hay imágenes que son muy distintas entre si. Comenta acerca de como crees que esta variabilidad podría afectar al rendimiento de un modelo de inteligencia artificial que intente clasificar estas imágenes.

1.2. Implementación y entrenamiento de una MLP (0.7 puntos)

Antes de implementar la MLP, responde las siguientes preguntas que te servirán de guía:

- Discute cómo la profundidad y el ancho (número de capas y neuronas por capa) de un MLP afectan su capacidad para aprender patrones complejos. ¿Cuáles son los desafíos relacionados con el aumento de la complejidad del modelo?
- Investiga y compara (a nivel teórico) al menos dos funciones de activación no lineales diferentes utilizadas en MLPs. ¿Cómo afectan estas funciones al tipo de decisiones que puede aprender la red? Considera aspectos como la saturación y la no linealidad.

Ahora debes implementar una MLP usando la librería `sklearn`. Debes hacer al menos 3 modelos con diferentes combinaciones de los hiperparámetros investigados previamente. Estos deben tener como mínimo 3 capas.

Luego entrena a los modelos en el set de train y haz un reporte de clasificación (es decir, evalúalos en las 4 métricas descritas anteriormente). En base a esos resultados decide cual es el mejor de los 3 y genera su matriz de confusión. Comenta sobre esta matriz.

Luego responde las siguientes preguntas:

Discute cómo el rendimiento de un MLP se ve afectado por el número de capas ocultas y el número de neuronas en cada capa. ¿Cuáles son las ventajas y desventajas de aumentar la complejidad de la arquitectura del MLP en términos de capacidad de aprendizaje y riesgo de overfitting?

1.3. Implementación y entrenamiento de una CNN (0.7 puntos)

Antes de la implementación investiga qué es una Red Neuronal Convolucional (CNN), y responde de manera breve las siguientes preguntas teóricas. Puedes encontrar información que te será útil en el siguiente **artículo**[3], el cual está orientado al uso de CNN's en imágenes y uso de la librería *Keras* para tareas de visión.

- ¿Qué es una operación convolucional? ¿Qué es un *kernel*? Utiliza estos conceptos para explicar el rol de las capas convolucionales en una CNN.
- ¿Cuál es el rol de las funciones de activación? ¿Y de las capas de *Max Pooling*?
- Quizás habrás notado que la mayoría de arquitecturas de CNN's utilizan una última capa conocida como *flatten layer*. ¿Cuál es su función? ¿Cuál es el rol de la función *softmax* en ella?

Ahora debes implementar una CNN utilizando el framework `keras`[4]. Luego investiga acerca de los hiperparámetros del modelo, y a continuación crea al menos 3 modelos con diferentes combinaciones de hiperparámetros. Estos deben tener como mínimo 3 capas convolucionales.

Luego repite el procedimiento de la pregunta anterior, es decir, entrena a los modelos en el set de train y haz un reporte de clasificación. En base a esos resultados decide cual es el mejor de los 3 y genera su matriz de confusión. Comenta sobre la matriz.

Luego responde las siguientes preguntas:

¿Qué modelo obtiene mejores resultados, el mejor MLP o la mejor CNN? Explica por qué una CNN tiende a funcionar mejor que un MLP para tareas relacionadas con imágenes. Argumenta tu respuesta basándote en las diferencias arquitectónicas entre ambas redes y cómo estas diferencias impactan en la capacidad de cada red para capturar características espaciales y patrones complejos.

1.4. Utilizar embeddings de `ResNet50`

A continuación lo que debes hacer es extraer los embeddings de las imágenes de hongos. Para esto, utiliza el código base entregado. Luego de este proceso, trabajaremos exclusivamente con estos embeddings para entrenar a los modelos.

Antes de continuar responde las siguientes preguntas **(0.3 puntos)**:

- ¿Qué es un embedding? Descríbelo con tus propias palabras.
- ¿Por qué puede ser beneficioso utilizar un extractor de embeddings en este tipo de tareas?
- Describe brevemente qué es una arquitectura `ResNet` y qué problemas intenta resolver en redes neuronales profundas. Explica qué es un embedding en este contexto y por qué los embeddings generados por `ResNet50` pueden ser útiles para clasificadores como SVM y MLP.

1.4.1. Entrenamiento de un Clasificador SVM con Embeddings de Imágenes (0.3 puntos)

Define y entrena un clasificador SVM utilizando la libería `sklearn`. Entrena al menos 3 modelos con diferentes combinaciones de hiperparámetros, usando como entrada los embeddings generados por ResNet50

Luego repite el procedimiento de las preguntas anteriores, es decir, entrena a los modelos en el set de train y haz un reporte de clasificación. En base a esos resultados decide cual es el mejor de los 3 y genera su matriz de confusión. Comenta acerca de esta matriz.

Luego responde las siguientes preguntas:

Del mejor clasificador SVM obtenido, ¿Que hiperparámetro puede haber favorecido a este SVM en este problema en particular? ¿Por qué?

[3] https://learnopencv.com/understanding-convolutional-neural-networks-cnn/
[4] https://keras.io/

1.4.2. Entrenamiento de un Clasificador MLP con Embeddings de Imágenes (0.3 puntos)

Define nuevamente los mismos MLP utilizados en la pregunta 1.1.1. y entrenalos usando los embeddings generados por ResNet50. Luego, haz un reporte de clasificación y decide cual es el mejor y genera su matriz de confusión. Comenta sobre la matriz.

Finalmente responde a las siguientes preguntas **(0.3 puntos)**:

- ¿Qué modelo obtiene mejores resultados, MLP sobre las imágenes o MLP sobre los embeddings? ¿Reemplazar las imágenes por embeddings generó un aumento en el rendimiento del modelo? Independiente de la respuesta anterior, ¿a qué puede deberse aquello? Explica cómo la transformación de las imágenes en embeddings ayuda a los modelos a capturar mejor las características importantes y teorizan por qué un modelo podría beneficiarse más que otro de esta representación.
- ¿Cuál de los modelos (SVM o MLP) tuvo un mejor rendimiento al usar los embeddings de `ResNet50`?. ¿A qué se debe esto? Justifica tu respuesta considerando cómo cada modelo maneja las características de alta dimensionalidad y las ventajas que uno podría tener sobre el otro.

Formato de Entrega

Entrega el cuaderno `.ipynb completo`, asegurándote de que todo el código esté comentado y sea fácil de seguir. Incluye tus respuestas a las preguntas teóricas y un análisis de los resultados obtenidos en el mismo notebook utilizando celdas de texto.

2. Flappy Bidoof (3 puntos)

En esta parte ocuparás aprendizaje reforzado (*Reinforcement Learning*) para entrenar una IA que sea capaz de aprender a jugar ***Flappy Bidoof***.

Figura 5: El protagonista del juego, **Bidoof**, sobre una plataforma

Flappy Bidoof es un juego en el que nuestro valiente protagonista, *Bidoof* salta de plataforma en plataforma en un intento por entrenar sus habilidades de volar y convertirse en el auténtico *Flappy Bidoof*. *Bidoof* avanza sin cesar por las columnas de plataformas que se avecinan mientras recolecta pizza y evita el fuego.

En esta pregunta entrenarás a un agente inteligente (*Bidoof*) para que aprenda a jugar utilizando el algoritmo *Q-Learning*. Existen tres filas e infinitas columnas. Cada paso, *Bidoof* avanza automáticamente hacia la derecha y debe decidir si saltar una fila más arriba, más abajo, o mantenerse en la actual. El juego acaba cuando *Bidoof* salta a un espacio sin plataforma y cae :(.

Para desarrollar este agente, debes implementar y entrenar un modelo de aprendizaje reforzado que permita a *Bidoof* decidir en cada paso debe cambiar a un nivel u otro, o mantenerse en el mismo.

A continuación se detalla más sobre el entorno para entrenar tu agente, en particular del espacio de estados, el espacio de acciones y la función de recompensas del entorno.

2.1. Espacio de estados

El espacio de estados en **Flappy Bidoof** se representa mediante un vector de 7 componentes enteras, donde la primera componente puede tomar valores entre 0 y 2 y el resto puede tomar valores entre 0 y 3, valores que indican distintos elementos del entorno. Este vector se estructura de la siguiente manera:

1. **Primera componente**: representa la **posición vertical del agente** (*Bidoof*), donde:
   - 0 indica que el agente está en el nivel más bajo,
   - 1 indica que está en el nivel medio,
   - 2 indica que está en el nivel superior.
2. **Siguientes 6 componentes**: brindan información sobre las próximas dos columnas en el camino, en grupos de tres:
   - **Las primeras 3 componentes** representan el contenido de la primera columna visible.

- **Las otras 3 componentes** representan el contenido de la segunda columna visible.

Cada componente de estas columnas puede tener los siguientes valores:

- **0**: la celda está vacía,
- **1**: hay una plataforma en la celda,
- **2**: hay una pizza en la celda.
- **3**: hay fuego en la celda.

La distribución de las celdas se ejemplifica a continuación:

Figura 6: Ejemplo de un estado del juego ***Flappy Bidoof***

En este caso el vector de estado sería: {1, 1, 2, 3, 0, 0, 1} Donde el azul es la posición intermedia, el rojo es la columna más cercana (1 es plataforma, 2 es pizza y 3 fuego) y el verde la plataforma sub siguiente.

2.2. Conjunto de acciones

Ahora, el espacio de acciones para el agente en **Flappy Bidoof** consta de 3 posibles valores, cada uno indicando una acción específica que el agente puede realizar:

1. **0**: Mantener la posición actual en el mismo nivel de altura.
2. **1**: Subir al nivel superior.
3. **2**: Bajar al nivel inferior.

2.3. Función de recompensa

La función de recompensa para el agente en **Flappy Bidoof** se define con tres posibles valores, dependiendo del resultado de la acción tomada:

- `r = 0`: El agente cae tras ejecutar la acción.
- `r = 1`: El agente aterriza exitosamente en una plataforma, lo cual le otorga una recompensa básica.
- `r = 6`: El agente aterriza en una plataforma y consigue una pizza, obteniendo una recompensa de 1 punto más un **bonus de 5 puntos** por recoger la pizza.
- `r = -1`: El agente aterriza en una plataforma pero esta tiene fuego, obteniendo una recompensa de 1 punto menos un **descuento de 2 puntos**.

2.4. Uso del entorno y archivos de la tarea

Los archivos contenidos en el código base son:

- `flappy_bidoof.py`: Código con la interfaz gráfica y funcionamiento general del juego contiene la clase FlappyBidoofEnv que modela el entorno con el que tu algoritmo interactuará. **No modificar**.
- `main.py`: Importa el entorno FlappyBidoofEnv y es donde debes realizar tu algoritmo. Se provee un esqueleto del código, pero puedes modificar este archivo como gustes. **Modificar**.

Como ya se mencionó anteriormente, el archivo sobre el cual debes trabajar e implementar *Q-Learning* es main.py. Este archivo contiene los siguientes métodos y clases:

- `FlappyBidoofEnv`: Esta clase corresponde al entorno del juego con el que tu algoritmo tiene que interactuar, se te entrega una instancia de este juego `env = FlappyBidoofEnv()` y a continuación se describen sus atributos y métodos más importantes.

Los métodos y atributos más importantes de FlappyBidoofEnv son los siguientes:

- `FlappyBidoofEnv.step()`: Función que modela la transferencia entre los estados del entorno. Recibe un número que representa la acción a realizar por el agente en el estado actual y retorna los siguientes valores:

```
next_state, reward, terminated, truncated, info = env.step(action)
```

  De estos los únicos que vas a necesitar son `next_state`, `reward` y `terminated`. Estos son respectivamente el siguiente estado tras ejecutar la acción, la recompensa producto de ejecutar la acción y `terminated` que indica si el estado alcanzado es un estado terminal.

- `FlappyBidoofEnv.reset()`: Este método reinicia el entorno en un estado aleatorio y retorna este estado inicial.

- `FlappyBidoofEnv.observation_space.nvec`: Retorna un vector con tantas componentes como el estado y por cada componente retorna la cantidad de valores que esta componente del estado puede alcanzar. En este caso retorna (3,4,4,4,4,4,4).
- `FlappyBidoofEnv.action_space.n`: Retorna la cantidad de acciones disponibles (tamaño del espacio de acciones). En este caso retorna 3.
- `FlappyBidoofEnv.render()`: Renderiza el estado actual del juego, esto es para que puedas ver a *Bidoof* aprendiendo a volar; no lo uses si vas a entrenar. En el código se usa de la siguiente manera:

```
env.render(window_size=winsize,done=done)
```

  Donde `window_size` es el tamaño de la ventana en pixeles y done indica si el juego terminó. No se espera que se modifique esta línea.

2.5. Actividades

A continuación realiza las siguientes actividades en tu archivo main.py

2.5.1. Actividad 1: Q-Table (0.3 puntos)

Inicializa la Q-table para tu algoritmo, esta tiene que ser flexible al tamaño de estados y cantidad de acciones, haciendo uso de los atributos entregados por el entorno.

2.5.2. Actividad 2: Implementar Q-learning (0.3 puntos)

Implementa Q-Learning para actualizar los valores de la Q-Table inicializada en el loop que se te entrega, tu implementación debe incluir los siguientes parámetros:

- Learning rate (α)
- Ratio de exploración (ϵ)
- Tasa de descuento (γ)

Por último tu implementación debe **registrar la Q-table** como un archivo .npy para poder testear el algoritmo luego.

2.5.3. Actividad 3: Hora de entrenar a tu agente (0.4 puntos)

Entrena tu agente hasta que notes una convergencia en el aprendizaje tomando como referencia el reward promedio de los últimos 1000 episodios.

2.6. Análisis teórico y resultados

2.6.1. Pregunta 0: Sobre Q-Learning (0.2 puntos)

1. ¿Qué representa la Q-table? ¿Qué significan sus entradas?
2. ¿Por qué Q-Learning se considera un algoritmo *off-policy*? ¿Qué ventaja tiene esto?

2.6.2. Pregunta 1: Análisis del Exploration Rate en Q-Learning (0.4 puntos)

1. ¿Cuál es la función del exploration rate en el algoritmo de Q-Learning y cómo afecta a la estrategia de exploración-explotación? Explica el rol del exploration rate (tasa de exploración) y analiza los efectos que tendría en el aprendizaje de este agente en casos extremos. ¿Qué ocurriría si este valor es muy bajo (cercano a 0) o muy alto (cercano a 1)? Reflexiona sobre cómo estos valores pueden impactar el proceso de exploración y explotación en el algoritmo.

2. ¿Qué valor de exploration rate te dio mejores resultados considerando la recompensa promedio y el tiempo de entrenamiento del agente? Presenta gráficos de reward promedio vs iteración para distintos valores de exploration rate y comenta qué valor de exploration rate dio los mejores resultados en términos de maximizar la recompensa promedio y minimizar el tiempo de entrenamiento, e incluye tu análisis sobre por qué crees que este valor fue el más efectivo.

2.6.3. Pregunta 2: Comprensión de la codificación del Espacio de Estados (0.3 puntos)

1. ¿Es el vector de estado que entrega el entorno la única forma de representar el estado del juego? ¿Podría el estado codificarse de otra forma? Explica.

2. ¿Es necesario incluir tanta información en el vector de estado? ¿Qué pasaría si el vector solo tuviese información de la columna inmediata? (es decir, si eliminamos las últimas tres componentes)

2.6.4. Pregunta 3: Evaluación de la Tasa de Descuento (Gamma) en el Aprendizaje (0.4 puntos)

1. ¿Cuál es el propósito de la tasa de descuento (gamma) en el algoritmo y qué valor te proporcionó los mejores resultados? Describe la función de la tasa de descuento (gamma) en Q-Learning y explica cómo este parámetro permite al agente equilibrar las recompensas inmediatas y futuras. Comenta qué valor de gamma te dio los mejores resultados en términos de rendimiento.

2. ¿Qué ocurriría en casos específicos si gamma tomara valores extremos como 0, 0.5 o 1? Realiza un análisis sobre el comportamiento del agente en estos casos, considerando estados ya avanzados del entrenamiento. Por ejemplo, describe situaciones específicas como qué decisión podría tomar el agente si una recompensa inmediata (como una "pizza."[en] una celda adyacente) se encuentra cerca, pero al observar más adelante no hay otra celda disponible para avanzar o hay fuego.

2.6.5. Pregunta 4: Limitaciones del Espacio de Estados del Agente (0.3 puntos)

¿Cuáles son las limitaciones actuales del agente en relación con su espacio de estados? Describe las restricciones en el espacio de estados del agente. Analiza por qué podría ser preferible utilizar solo las dos columnas siguientes en lugar de una representación más extensa. Menciona posibles maneras de mejorar estas limitaciones y los problemas que podrían surgir con estas soluciones, considerando tamaño del espacio de estados.

2.6.6. Pregunta 5: Análisis y Presentación de Resultados (0.4 puntos)

1. Presenta un gráfico de recompensa promedio vs. iteración que muestre la evolución del rendimiento del agente durante el entrenamiento. ¿En qué punto observas que el agente comienza a aprender una política estable?

2. Evalúa la política determinista aprendida por *Bidoof*. Para esto, cada N episodios (definido según tu criterio, por ejemplo 100) haz una pausa en el entrenamiento y extrae la política determinista/greedy (es decir, considerando épsilon=0) aprendida por tu agente hasta el momento. Realiza K ejecuciones (definido según tu criterio, por ejemplo 10) con esa política y guarda la recompensa promedio. Al finalizar el entrenamiento, muestra un gráfico de la evolución de esa política. Explica las diferencias que encuentres entre la recompensa promedio obtenida en esta evaluación y la recompensa promedio que se observa durante el entrenamiento.

2.6.7. ¡Bonus! (0.5 puntos)

Dada una serie de partidas con diferentes *seed* para el juego se van a poner a prueba los agentes y los cinco mayores puntajes promedio del curso entre las partidas obtendrán un bonus. Para poder acceder al bonus se tiene que registrar la Q-Table que se quiere presentar para el bonus en el repositorio, con el nombre `Bonus_TuUsuarioDeGithub.npy` y esta debe ser de una dimensión de 12288x3.

Comentarios Finales

Como podrás notar, buena parte de esta tarea involucra respuestas de desarrollo escrito, donde debes transparentar tu razonamiento y explicar las decisiones que tomas. Por este motivo, para que una respuesta se considere correcta, debes tener cuidado de fundamentar apropiadamente lo que digas, aludiendo a referencias confiables y a la documentación de las librerías que utilices. Por ejemplo, si te pedimos explicar un hiperparámetro en particular, o calcular una métrica de desempeño, se espera que demuestres un dominio general de lo que hace dicho hiperparámetro, o de la información que entrega la métrica solicitada. Para esto, es esperable que busques recursos (libros, artículos, documentación oficial, etc.) para fundamentar tus respuestas, donde debes indicar claramente la fuente de tal recurso.

Al mismo tiempo, es posible que al intentar ejecutar operaciones costosas (como entrenar un modelo sobre un conjunto de datos), la ejecución sea más lenta de lo que esperas. Esto es un escenario cotidiano al trabajar con modelos de aprendizaje de máquina, de modo que se espera que seas capaz de lidiar con tales situaciones, y que transparentes y fundamentes las decisiones que tomes en el proceso.

Política de Integridad Académica

Los/as estudiantes de la Escuela de Ingeniería de la Pontificia Universidad Católica de Chile deben mantener un comportamiento acorde a la Declaración de Principios de la Universidad. En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los/as estudiantes que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada estudiante conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Docencia de la Escuela de Ingeniería.

Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un/a estudiante para los efectos de la evaluación de un curso debe ser hecho **individualmente** por el/la estudiante, **sin apoyo en material de terceros**. Por "trabajo" se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros.

En particular, si un/a estudiante copia un trabajo, o si a un/a estudiante se le prueba que compró o intentó comprar un trabajo, **obtendrá nota final 1.1 en el curso** y se solicitará a la Dirección de Pregrado de la Escuela de Ingeniería que no le permita retirar el curso de la carga académica semestral.

Por "copia" se entiende incluir en el trabajo presentado como propio, partes hechas por otra persona. En caso que corresponda a "copia" a otros estudiantes, la sanción anterior se aplicará a todos los involucrados. En todos los casos, se informará a la Dirección de Pregrado de la Escuela de Ingeniería para que tome sanciones adicionales si lo estima conveniente.

También se entiende por copia extraer contenido sin modificarlo sustancialmente desde fuentes digitales como Wikipedia o mediante el uso de asistentes inteligentes como ChatGPT o Copilot. Se entiende que una modificación sustancial involucra el análisis crítico de la información extraída y en consecuencia todas las modificaciones y mejoras que de este análisis se desprendan. Cualquiera sea el caso, el uso de fuentes bibliográficas, digitales o asistentes debe declararse de forma explícita, y debe indicarse cómo el/la estudiante mejoró la información extraída para cumplir con los objetivos de la actividad evaluativa.

Obviamente, está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, **siempre y cuando se incluya la referencia correspondiente**.

Lo anterior se entiende como complemento al Reglamento del Estudiante de la Pontificia Universidad Católica de Chile (`https://registrosacademicos.uc.cl/reglamentos/estudiantiles/`). Por ello, es posible pedir a la Universidad la aplicación de sanciones adicionales especificadas en dicho reglamento.

Compromiso del Código de Honor

Este curso suscribe el Código de Honor establecido por la Universidad, el que es vinculante. Todo trabajo evaluado en este curso debe ser propio. En caso que exista colaboración permitida con otros/as estudiantes, el trabajo deberá referenciar y atribuir correctamente dicha contribución a quien corresponda. Como estudiante es un debe conocer el Código de Honor (`https://www.uc.cl/codigo-de-honor/`).