PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
Facultad de Ingeniería
IIC2613 Inteligencia Artificial
2025-1

Tarea 5
Aprendizaje Reforzado y Aprendizaje Profundo

Fecha de entrega: Viernes 27 de Junio 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 viernes 27 de junio 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. DCCompresor: Compresor de texto usando Redes Neuronales

Introducción

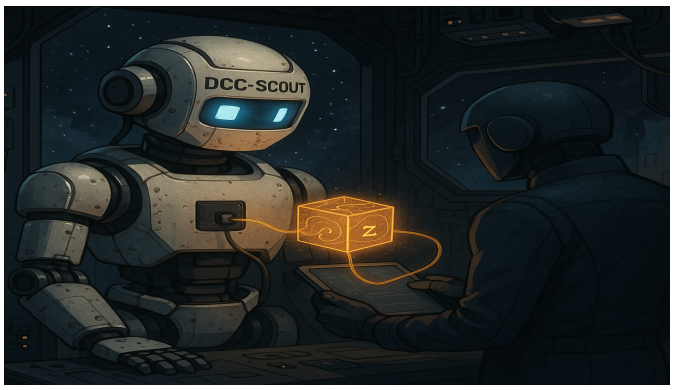

Figura 1: DCC-Scout y el equipo de desarrollo

En una operación secreta detrás de líneas enemigas, el agente DCC-Scout tiene una instrucción urgente, realizar la transmisión de un mensaje con información relevante que no debe ser interceptada ni comprendida por otros en ninguna circunstancia. El contenido del mensaje posee información que puede comprometer toda su misión si cae en manos equivocadas.

Para ello debe proteger la información a toda costa, pero no puede recurrir a cifrados comunes, ya demasiado conocidos y vulnerables a ataques de intercepción. Para ello, se requiere la implementación de un sistema de codificación y compresión basado en inteligencia artificial, es decir, un modelo entrenado para transformar el mensaje en una secuencia cifrada ilegible para cualquier persona sin el modelo adecuado.

Este sistema no busca solo reducir el tamaño del mensaje, sino ocultar su verdadero significado en una forma codificada solamente interpretable por quienes posean el decodificador correcto. La codificación generada por el este modelo debe ser prácticamente indistinguible del ruido para los demás, pero adecuada para contener toda la información necesaria para reconstruir de la mejor manera posible el mensaje original.

Tu misión, como parte del equipo de comunicaciones seguras, es poder diseñar y poner en marcha un prototipo funcional de este sistema de compresión y descompresión basado en redes neuronales. Debe ser robusto, compacto y capaz de reconstruir el mensaje original con la mayor fidelidad posible, incluso si la secuencia comprimida sufre distorsiones durante su transmisión. La seguridad de la operación depende de ello.

1.1. Compresión basada en Modelos de aprendizaje profundo (Total: 3 puntos)

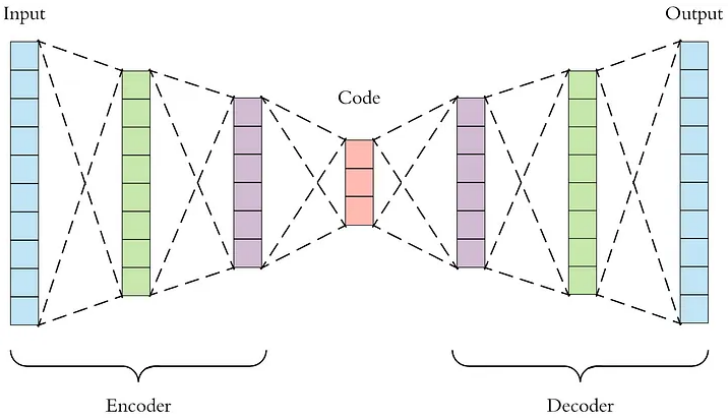

Figura 2: Forma del compresor y descompresor.
[1]

En esta implementación, se busca desarrollar un sistema de compresión utilizando un modelo de aprendizaje profundo. Se espera que el modelo procese secuencias completas de texto y aprenda a representarlas mediante un vector z, el cual contiene toda la información necesaria para reconstruir el texto lo más similarmente posible al original.

Este vector z, de tamaño fijo (por ejemplo, $K \times 1$ dimensiones, con K un número entero), actúa como una forma comprimida del mensaje original, que considera tanto las frecuencias de aparicion de las palabras como las relaciones entre ellas. Se espera que el modelo aprenda las estructuras y patrones del lenguaje para generar representaciones eficientes de este vector. Es decir, si mi frase de entrada es *thank you*, se espera que el resultado sea lo más parecido posible a la misma frase original, pero determinado a partir de la decodificación/descompresión del vector z.

1.2. Conjunto de datos

Para entrenar sus modelos, pueden usar cualquier conjunto de datos de texto disponible, donde solo consideraremos las frases en inglés en código ASCII. Por ejemplo, pueden extraer frases de **Tatoeba**, **20 Newsgroups**, o **Reuters News dataset** entre otros. Se recomienda que el conjunto de datos sea lo más extenso (entre 200000 a 400000 frases como mínimo) y lo más diverso posible, con el fin de capturar la mayor cantidad de palabras.

1.3. Preguntas a responder

- **Autoencoders**: investigue sobre los modelos autoencoders, mencione a grandes rasgos cómo es su estructura y responda: ¿Por qué son útiles para comprimir y descomprimir datos? (extensión máxima 7 lineas)
- **Tokenización**: investigue acerca de la tokenización de los textos. Describa brevemente el concepto y plantee un esquema de tokenización para sus datos. En este sentido, justifique el esquema que utilizará, ya sea por palabras o por caracteres. (extensión máxima 7 lineas)
- **Funciones de pérdida**: investigue sobre las funciones de pérdida y en particular sobre: `Cross Entropy Loss`. Explica cómo funciona y reflexiona sobre su utilidad en este contexto. (Extensión máxima 9 líneas).
- **Dropout**: discuta sobre la utilidad de utilizar dropout en este contexto. Indique además en cuales capas de un autoencoder utilizaría dropout y en cuales no. (Extensión máxima 10 líneas).

[1] Imagen extraída de Medium

1.4. Creando un Modelo de aprendizaje profundo para compresión y descompresión de texto

Para esta parte, se espera que entrene un modelo que conste de dos partes: la primera, el compresor, dado un texto de entrada X, debe comprimirlo en un vector Z de dimensionalidad Y. Este será el modelo compresor. La segunda parte del modelo, el descompresor, dado como entrada el vector Z, debe ser capaz de descomprimirlo en el vector X (o lo más similar posible). Se espera que el vector Z sea lo más compacto posible sin tener pérdidas significativas de información.

1.4.1. Consideraciones

1. Se espera que se aplique la función de pérdida investigada anteriormente. La arquitectura del modelo queda a su criterio. Es decir, puede utilizar cualquiera de las arquitecturas vistas en clases, o derivadas de ellas, mientras justifique la decisión tomada (considere en esto los hiperparámetros a seleccionar).

2. Para el optimizador, se recomienda usar el optimizador Adam.

3. Se espera que se utilice el esquema de tokenización de las frases planteado anteriormente.

4. Se recomienda utilizar un vocabulario en base a los datos que posee, es decir, las palabras que serán consideradas en los datos; como sugerencia, sería bueno considerar una frecuencia mínima de apariciones de palabras para agregar o no a su vocabulario una palabra. Un vocabulario muy extenso requiere mucho computo y uno muy pequeño puede no ser adecuado. A su vez, también es recomendable que investigue sobre el uso de caracteres especiales como adición al vocabulario, tales como el $< PAD >$, $< UNK >$, $< SOS >$, $< EOS >$; no es necesario que los utilicen todos, pero sí es recomendable.

5. Considere realizar una separación entre datos de train y test.

6. Dado los límites computacionales que posee Colab, considere limitar el tamaño de las frases a un tope máximo, por ejemplo, un máximo de 20 a 30 palabras por frase.

7. La dimensionalidad Y del vector Z queda a su criterio, pero justifique su elección y el porqué de ese tamaño.

1.5. Análisis del Modelo de Aprendizaje Profundo

- **Hiperparámetros**: Explique la elección de hiperparámetros que realizó; entiéndase por hiperparámetros elecciones como épocas de entrenamiento, longitud máxima de las frases, número de capas, tamaño de las capas, etc.

- **Resultados**: Presente una muestra de frases originales, vector de compresión Z y descompresiones hechas por el modelo, determine si su modelo tiende a fallar más en frases cortas o largas, y explique por qué razón podría ocurrir esto.

- **Exactitud**: Determine la cantidad de aciertos que tiene su modelo para predecir frases por medio de la cantidad de tokens en la posición correcta que el modelo entregó, **obtenga al menos un 50 % de accuracy a nivel de token por posición**.

  Ej: Si nuestra tokenización de la frase original corresponde a `[1, 1, 23, 5, 102, 234]` y la tokenización entregada al predecir por el modelo es `[1, 23, 4, 5, 102, 234]`, entonces:

  - Tokens coincidentes: posiciones 0, 3, 4 y 5.
  - Total de tokens: 6
  - Accuracy $= \frac{4}{6} \times 100 = 66.67\,\%$

También calcule la cantidad de aciertos por frecuencia de palabras entre la frase original y la frase generada, sin importar el lugar en el que esté un token.

Ej: Si nuestra tokenización de la frase original corresponde a `[1, 1, 23, 5, 102, 234]` y la tokenización entregada por el modelo es `[1, 23, 4, 5, 102, 234]`, entonces:

Nuestro Accuracy por frecuencia es del $\frac{5}{6} \times 100 = 83\,\%$, pues aparecen 5 de las palabras originales, pero en este caso algunas están en un lugar incorrecto o faltantes.

Comente sobre sus resultados y qué pueden indicar los valores obtenidos sobre cómo el modelo predice las palabras. (extensión máxima 3-4 líneas)

- **Errores de palabras**: Comente sobre cuáles son las palabras que más errores presenta su modelo, y comente a qué podría deberse esto (frecuencia, ambigüedad, falta de datos, etc.). (extensión máxima 5 líneas)
- **Compresor**: Comente si este tipo de modelos es bueno para realizar compresión de texto. Con base en lo que investigo y determinó con sus resultados, decida si serviría o no para este propósito, ya sea con mejores ajustes, mayor capacidad de cómputo, entrenamiento, más memoria, etc. (extensión máxima 5-7 líneas)
- **Seq2Seq**: Investigue sobre las redes Seq2Seq, mencione diferencias y similitudes sobre el modelo que usted desarrolló y este tipo de redes. (extensión máxima 5-6 líneas)

1.6. Algunas consideraciones adicionales

- **Uso de Colab**: Dado que el entrenamiento de redes neuronales requiere una considerable capacidad de cómputo (particularmente GPU o TPU), y entendiendo que no todos los estudiantes disponen del hardware necesario en sus equipos personales, se recomienda el uso de Google Colab como entorno de desarrollo de su tarea.

  1. Acceder a **Google Colab**[2]
  2. Abrir el archivo `.ipynb` correspondiente a la tarea (puede subirse manualmente o abrirse desde Google Drive).
  3. Ir al menú **Entorno de ejecución** → **Cambiar tipo de entorno de ejecución**.
  4. En la opción *Acelerador de hardware*, seleccionar **GPU**.
  5. Hacer clic en **Guardar** para aplicar los cambios.
  6. Finalmente, seleccionar **Entorno de ejecución** → **Ejecutar todo** para comenzar.

- **Librerías Recomendadas**: Para el desarrollo de esta tarea se recomienda el uso de la librería **PyTorch**, debido a su sintaxis clara, flexibilidad en la construcción de modelos y mensajes de error comprensibles, lo que facilita el proceso de depuración y aprendizaje. No obstante, pueden utilizar otras librerías de aprendizaje profundo como **Keras** si así lo prefieren, siempre que cumplan con los requisitos funcionales de la tarea.

1.7. Entregables

Un notebook de Jupyter (**.ipynb**) que contenga las respuestas a las secciones anteriores. **Debe ser entregado con todas sus celdas de código ejecutadas, de lo contrario no será corregido.** Recuerda que toda decisión debe ser justificada, y que cualquier resultado debe ser comentado. Asimismo, recuerda comentar el código explicando que hace este.

[2]Los estudiantes que deseen utilizar su propio computador para el desarrollo de la tarea son libres de hacerlo, siempre y cuando cumplan con los requisitos establecidos.

2. DCCPong: Aprendizaje Reforzado (Total: 3 puntos)

En esta parte ocuparás aprendizaje reforzado (*Reinforcement Learning*) para entrenar una IA que sea capaz de aprender a jugar al clásico juego Pong por sí misma:

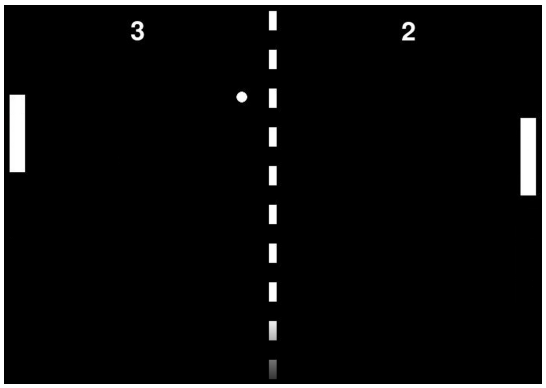

Figura 3: GUI del juego Pong

En tu repositorio se te da el código del juego Pong, pero el pádel (paleta móvil que controla el jugador) se mueve de manera aleatoria. Deberás implementar el algoritmo *Q-Learning* para que pueda moverse de forma "inteligente", respondiendo la mayor cantidad de pelotas posibles.

La interfaz usada para entrenar al agente es diferente a la que se utiliza para jugar el juego. Para simplificar el proceso de entrenamiento, el pádel no juega contra otro pádel, sino que juega solo, recibiendo pelotas lanzadas en posiciones aleatorias desde el fondo de la cancha.

Los archivos contenidos en el código base son:

- `PongAI.py`: Código con la interfaz gráfica y funcionamiento general del juego. **No modificar**.
- `QAgent.py`: Este archivo contiene el modelo de un agente que se mueve de forma aleatoria. Es aquí donde deberás trabajar y aplicar tus conocimientos de aprendizaje reforzado.

Como ya se mencionó anteriormente, el archivo sobre el cual debes trabajar e implementar *Q-Learning* es `QAgent.py`[3]. Este archivo contiene la clase `Agent` con los siguientes métodos:

- `__init__()`: Este método inicializa los atributos del agente. En el caso del agente aleatorio, solo se inicializan en 0 las partidas jugadas por el agente.
- `get_state(game)`: Este método consulta al juego por el estado actual del agente y lo retorna como un vector o tupla de 5 dimensiones, donde cada una de las entradas representa la siguiente información (en orden):
  1. **Velocidad en el eje Y:** Toma un valor entero entre -5 y 5 correspondiente a la velocidad actual de la pelota en el eje Y. Los valores negativos indican un movimiento hacia arriba de la cancha (se redondea al entero más cercano).

[3]Este archivo se encuentra muy bien comentado, por lo que se recomienda leerlo como parte del enunciado.

2. **Proximidad al pádel:** Toma un valor entero entre 0 y 5 que representa la distancia entre el pádel y el cuadrante en el que se encuentra la pelota (son 6 cuadrantes), tal como se muestra en la figura:

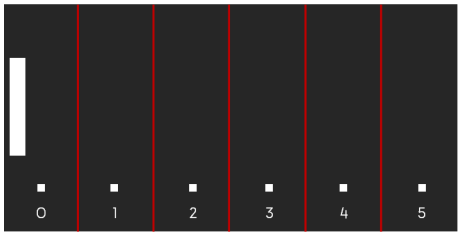

Figura 4: Proximidad del pádel y cuadrante de la pelota

3. **Altura del balón respecto al extremo superior del pádel:** Es un entero que puede tomar los valores 0, 1, 2 y 3. Si la pelota se encuentra arriba del extremo superior del agente, entonces toma el valor 0 si está a menos de un pádel de distancia de dicho extremo, o 1 si está más lejos. Por el contrario, si la pelota está debajo del extremo superior del agente, toma el valor 2 si está a menos de un pádel de distancia y el valor 3 si está más lejos.
4. **Altura del balón respecto al extremo inferior del pádel:** Es un entero que puede tomar los valores 0, 1, 2 y 3. Si la pelota se encuentra arriba del extremo inferior del agente, entonces toma el valor 0 si está a menos de un pádel de distancia de dicho extremo, o 1 si está más lejos. Por el contrario, si la pelota está debajo del extremo inferior del agente, toma el valor 2 si está a menos de un pádel de distancia y el valor 3 si está más lejos.

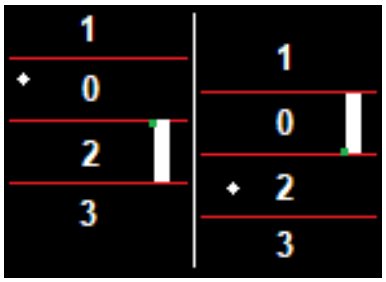

Figura 5: Referencia respecto al extremo superior (izq.). Referencia respecto al extremo inferior (der.)

5. **Número de rebotes:** Es un entero correspondiente al número de rebotes que ha dado la pelota desde el golpe del rival. Por físicas del juego, se encuentra entre 0 y 2.

- `get_action(state)`: Este método recibe el estado del agente y retorna un entero representando la acción que debe tomar el agente. Las acciones posibles son:
  - `0`: Moverse hacia arriba.
  - `1`: No moverse.
  - `2`: Moverse hacia abajo.

  Nota: En el caso del agente aleatorio, este no utilizará la información que entrega el estado y retornará una acción al azar.

Fuera de esta clase, sólo existe la función `train()`. Esta función es la encargada de entrenar al agente y es la que se llama al correr el archivo de ejecución principal.

Actividad 1: Comprendiendo los hiperparámetros (0.5 puntos)

Caso: Dos estudiantes entrenan pádeles de Pong con configuraciones diferentes. El Agente A usa learning rate = 0.1 y discount factor = 0.9, mientras que el Agente B usa learning rate = 0.8 y discount factor = 0.1. Después de 10000 episodios, el Agente A muestra mejoras graduales y consistentes, pero el Agente B tiene oscilaciones abruptas en su rendimiento, alternando entre mejoras y caídas bruscas.

1. Investiga la ecuación de actualización $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max Q(s',a') - Q(s,a)]$. ¿Cómo explica matemáticamente esta ecuación las diferencias observadas entre ambos agentes? Analiza el impacto específico de $\alpha = 0.8$ versus $\alpha = 0.1$ en la magnitud de las actualizaciones.

2. Investiga cómo funciona la exploración ϵ-greedy y su decaimiento. Si ambos agentes comienzan con $\epsilon = 1.0$ y decay $= 0.995$, ¿en qué episodio aproximadamente cada uno tendría $\epsilon = 0.1$? ¿Cómo interactúa esto con sus diferentes learning rates?

3. ¿Por qué el Agente B experimenta oscilaciones mientras que el Agente A mejora gradualmente? ¿Qué hiperparámetro específico causa esta inestabilidad y cómo?

4. El Agente A con $\gamma = 0.9$ planifica más hacia el futuro que el Agente B con $\gamma = 0.1$. En el contexto de Pong, ¿qué ventajas específicas tendría el Agente A al anticipar la trayectoria de la pelota varios movimientos adelante?

5. Si quisieras combinar lo mejor de ambos agentes (la velocidad inicial del B y la estabilidad del A), ¿qué configuración de hiperparámetros propondrías y por qué? Justifica cómo cada parámetro contribuiría al objetivo deseado.

Actividad 2: Implementando Q-Learning (0.75 puntos)

Basándote en lo visto en clases y ayudantías, implementa el algoritmo *Q-Learning*[4] para el agente del juego Pong, esto es, el pádel. Para esto deberás realizar los siguientes pasos [5] (recuerda comentar todo lo que hagas en tu código):

1. Inicializar la *Q-Table* del agente en el método `__init__` con sus dimensiones correctas **(0.125 puntos)**.

2. Modificar el método `get_action` para que el agente sea capaz de explorar el estado actual o explotar la mejor acción conocida hasta el momento **(0.25 puntos)**.

3. Actualizar la *Q-Table* una vez que se ejecute la acción seleccionada por el agente **(0.2 puntos)**.

4. En caso de terminar una partida, actualizar el *exploration rate* del agente **(0.175 puntos)**.

IMPORTANTE Y OBLIGATORIO

Para facilitar la corrección de tu tarea, te pediremos subir la *Q-Table* de tu agente ya entrenado como un archivo en formato `<usuario_github>.csv` (por ejemplo si el usuario es `ia2025`, el archivo debe llamarse `ia2025.csv`), sin *headers* (filas/columnas con títulos o nombres). Este debe estar dentro del directorio `DCCPong`.

El archivo .csv debe poseer 8 columnas y 3168 filas. Las primeras 5 columnas corresponden a los posibles valores de las 5 dimensiones de los estados en el orden que se mencionan en este enunciado. Por otra parte, las siguientes 3 columnas corresponden al *Q-Value* de las acciones posibles del pádel también en el orden del enunciado. De esta forma, cada fila representa un posible estado y los valores de las acciones posibles para ese estado. Una fila de este archivo se podría ver así:

[4] Si tienes dudas con *Q-Learning*, puedes revisar este artículo.

[5] Se recomienda revisar el código base, específicamente los `#IMPLEMENTAR` que hay en el código

$$[-4,\ 1,\ 1,\ 0,\ 1,\ 1.9878,\ 0.243,\ 99.346]$$

Este formato también es el mismo requerido para poder participar en el bonus, así que estás invitado/a/e a participar.

Actividad 3: Experimentación y Análisis del Agente (1.5 puntos)

Para esta sección, deberás realizar cuatro experimentos distintos modificando los hiperparámetros del agente. Para cada experimento, debes **generar evidencia** (gráficos, tablas de datos) y usarla para fundamentar tus respuestas.

Experimento 1: El Agente "Miope" vs. el Agente "Previsor" (0.375 puntos)

El código inicial utiliza un `DISCOUNT_RATE = 0`, lo que crea un agente "miope" (solo le importa la recompensa inmediata). Para este experimento, deberás seguir los siguientes pasos:

1. Realizar dos entrenamientos: uno con el valor original (`DISCOUNT_RATE = 0`) y otro con un valor alto que incentive al agente a ser **"previsor"** (por ej., `DISCOUNT_RATE = 0.95`). Entrena cada uno por un número significativo de episodios (ej. 50.000).
2. **Genera un gráfico** que muestre el `mean_score` (puntaje promedio) a lo largo del tiempo para *cada entrenamiento* en la misma figura.
3. Analizando el gráfico, responde: ¿Qué agente aprende más rápido al principio? ¿Cuál alcanza un mejor rendimiento máximo a largo plazo? ¿Cómo la evidencia visual de tu gráfico demuestra la diferencia fundamental entre un agente que busca recompensas inmediatas y uno que planifica para el futuro? En base a esto, ¿qué valor de `DISCOUNT_RATE` elegirías?

Experimento 2: El Dilema de la Velocidad vs. la Estabilidad (0.375 puntos)

El `LR` (Learning Rate) no solo afecta qué tan rápido aprende el agente, sino también qué tan estable es su conocimiento. Para este experimento, deberás seguir los siguientes pasos:

1. Realiza un experimento comparando un `LR` muy bajo (ej. `0.01`) contra un `LR` muy alto (ej. `0.7`).
2. Para cada caso, registra el `record` (puntaje récord) y el `mean_score` cada 100 juegos. **Presenta tus resultados en una tabla comparativa.**
3. Basado en la tabla, compara el proceso de aprendizaje del agente con tasas de aprendizaje (LR) bajas y altas. Describe la velocidad y las características de su rendimiento (p. ej., estable, errático, estancamiento) en cada escenario. Finalmente, selecciona el valor de LR que consideres óptimo y justifica tu elección basándote en el compromiso (*trade-off*) observado entre la velocidad de convergencia y la estabilidad del rendimiento.

Experimento 3: Saboteando el Aprendizaje (0.375 puntos)

El balance entre explorar el entorno y explotar el conocimiento adquirido es fundamental. En este experimento, investigaremos qué pasa cuando ese balance se rompe. Para ello, deberás seguir los siguientes pasos:

1. Diseña y ejecuta dos experimentos "fallidos":
   a) **Explotación Prematura:** Modifica el `EXPLORATION_DECAY_RATE` a un valor muy alto (ej. `0.05`) para que el agente deje de explorar casi inmediatamente.
   b) **Exploración Eterna:** Modifica `MIN_EXPLORATION_RATE` para que sea igual a `MAX_EXPLORATION_RATE` (ej. `0.8`, forzando al agente a tomar una acción aleatoria el 80 % de las veces, para siempre.

2. Describe el comportamiento observado en los escenarios de "Explotación Prematura" y "Exploración Eterna". Evalúa la calidad de las políticas aprendidas y fundamenta cómo el balance entre exploración y explotación en cada caso explica el rendimiento final del agente.

3. Explica cómo los resultados de estos dos experimentos fallidos demuestran por qué es crucial tener una tasa de exploración que **decaiga gradualmente** con el tiempo.

Experimento 4: Entrenando a tu Agente Ideal (0.375 puntos)

Hemos descubierto que los hiperparámetros del agente dictan su desempeño (ya sea para bien o para mal). Para este experimento, diseñaremos y entrenaremos a nuestro "agente ideal". Para ello, deberás seguir los siguientes pasos:

1. Selecciona y justifica brevemente la configuración de hiperparámetros que consideres óptima y reentrena al agente con ella.

2. **Genera un gráfico** que muestre el `mean_score` (puntaje promedio) a lo largo del tiempo del entrenamiento.

3. Reflexiona sobre el rendimiento de tu agente. Considera si un agente "perfecto" es un objetivo realista y describe qué acciones específicas tomarías para mejorar aún más los resultados obtenidos, justificando tu propuesta.

Actividad 4: Nueva política de recompensas (0.25 puntos)

Actualmente, el juego otorga recompensas al agente calculando la posición en donde terminará la pelota; asignando 1 punto si se acerca hacia esta posición (o se encuentra en ella) o restando 1 punto si se está alejando de esta.

Para esta actividad deberás diseñar (no es necesario implementar) una política de recompensa distinta a la implementada en el archivo `Pong.py` para poder entrenar al agente y responder:

- ¿Por qué crees que sería una buena forma de recompensar al agente? Argumenta.

Bonus: DCCampeonato IIC2613 2025-1

Para finalizar el semestre de forma especial, celebraremos la **segunda edición** del DCCampeonato IIC2613, en donde puedan poner a prueba sus agentes de aprendizaje reforzado y competir con sus demás compañeros de curso en un torneo de pádel virtual. Quienes ganen este evento **podrán ganar hasta 10 décimas** en su tarea.

Los entregables para participar del campeonato son:

1. La *Q-Table* del agente entrenado, en el mismo formato requerido para la entrega y corrección de la tarea (debe llamarse `<usuario_github>.csv`)
2. Una foto de perfil en el *root* de su repositorio (debe llamarse `<usuario_github>.png`) para que puedan identificar, en vivo, cuál es su agente. Puede ser lo que ustedes quieran (mientras no sea contenido inapropiado), ¡sean creativos!

Ambos deben estar dentro del directorio `DCCPong`.

Para participar, deberán inscribirse en este formulario[6] (igualmente se enviará mediante un anuncio de Canvas). Contamos con su motivación y participación para hacer de esta segunda edición una instancia pedagógica y recreativa que pueda mantenerse a futuro.

[6] Link del formulario: https://forms.gle/hYkee4hJcqMQmJ6H9

Referencias

[1] Tec With Tim. (2022, 15 de febrero). *Make Pong With Python!* [Video]. YouTube. Recuperado el 18 de mayo del 2022 en: https://www.youtube.com/watch?v=vVGTZlnnX3U