PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
Facultad de Ingeniería
IIC2613 Inteligencia Artificial
2023-1

Tarea 4

Aspectos generales

Formato y plazo de entrega

El formato de entrega es un archivo con extensión .ipynb. Es importante que a la hora de entregar, tu notebook se encuentre **con las celdas ejecutadas**, de lo contrario no será corregido. Las respuestas a las preguntas teóricas deben incluirse en celdas de texto adyacentes al código que utilices, de manera tal que el texto y las celdas de código sigan una estructura lógica que evidencie tu trabajo. El lugar de entrega es en el repositorio de la tarea, en la *branch* por defecto, hasta el **16 de junio a las 23:59**. 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.

DCCat vs Mouse

En esta tarea combinarán los tópicos de aprendizaje reforzado y redes neuronales, mediante el entrenamiento de dos agentes que participan de un juego de cacería entre un gato y un ratón. En este, un gato "cazador"(representado en la interfaz por el color rojo) debe perseguir y atrapar a un ratón (representado por el color verde) que debe huir de él. Ambos agentes deben moverse por un mapa con obstáculos, como el de la siguiente figura:

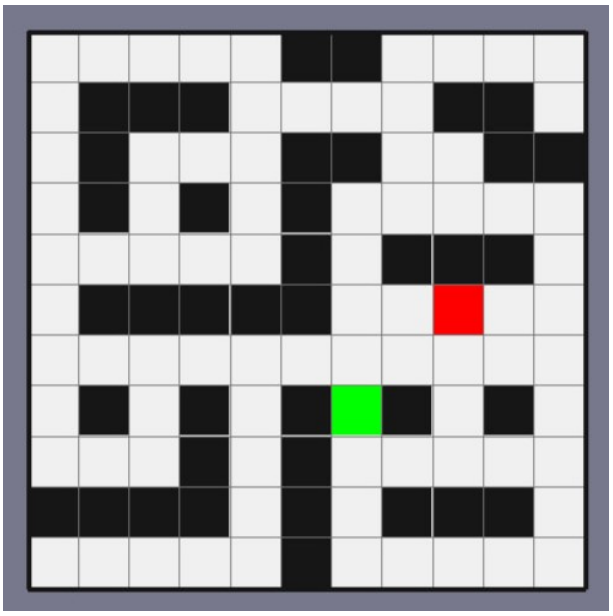

Figura 1: Mapa de juego: En rojo tenemos al cazador, en verde al agente perseguido

Las casillas negras evitarán el paso de cada agente, lo que implica que el movimiento de este hacia una pared mantendrá al personaje en su casilla actual. Por otro lado, las casillas blancas representan espacios libres dentro del mapa por los que cada agente puede transitar.

Para cada instante de tiempo, cada agente deberá decidir qué acción tomar de un conjunto de 5 acciones distintas representadas por un número entero entre 0 y 4.

Acción	**Movimiento**
0	Arriba
1	Abajo
2	Izquierda
3	Derecha
4	Nada

Cuadro 1: Movimientos del agente y el índice de acción asociado

Código Entregado

La tarea se puede separar en tres módulos distintos que modelan distintos componentes dentro de su implementación. Cada archivo se encuentra exhaustivamente comentado para guiarte en su desarrollo.

1. Código general

- **test.py**: archivo para testear los agentes. Aquí deberás modificar solo las líneas que se indican al principio del archivo para poder usar tu agente.
- **chase_game.py**: define toda la lógica del juego. **No modificar.**
- **agents/baseline.py**: archivo que contiene los agentes base para que puedas entrenar tus modelos haciéndolos jugar contra estos. **No modificar.**
- **utils.py**: archivo que contiene una clase para representar cada celda del mapa. Además contiene funciones útiles que se usan en el resto de archivos. **No modificar.**

2. Código de Redes Neuronales:

El código para esta parte de la tarea consiste en dos archivos:

- **train_neural_agent.ipynb**: Jupyter Notebook que contiene lo necesario para que puedas crear, entrenar y exportar las redes neuronales que modelen el comportamiento de cada agente (entrenarás una red distinta para cada uno).
- **agents/neural.py**: contiene las clases `NNCat` y `NNMouse` que recibirán como argumento los directorios a las redes generadas en el notebook para que puedan controlar al agente del gato y el ratón respectivamente.

3. Código de Aprendizaje Reforzado

Los módulos donde se trabajará con Reinforcement Learning son:

- **train_reinforced_agent.py**: se encarga de entrenar este agente y no debe ser modificado.
- **reinforced.py**: archivo sobre el cual debes trabajar e implementar *Q-Learning*. Este archivo contiene la clase `ReinforcedAgent` y sus métodos:
  - **__init__()**: este método inicializa los atributos del agente. En el caso del agente, su posición inicial y q_table son dadas como atributos.
  - **get_action**: este método obtiene la acción que efectuará el agente dado un estado de juego actual. Retorna el movimiento (valor entre 0 y 4 inclusive)
  - **get_reward**: este método se encarga de calcular la recompensa para la acción real realizada por el agente. Retorna la recompensa
  - **get_update_policy**: se encarga de actualizar la politica empleada, en este caso actualiza el valor asociado al par estado-acción de la q_table. No retorna, solo actualiza el par estado-accion en la q_table
  - **get_update_exploration**: este método se encarga de actualizar la tasa de exploración del agente. No retorna, solo actualiza el **self.exploration_rate**

Parte 1: implementación de Red Neuronal (2,5 ptos.)

En esta actividad tendrás que realizar dos veces cada cosa descrita, una para cada uno de los agentes.

Actividad 1.1: creación de el vector de etiquetas

En esta actividad deberás crear un vector de etiquetas para una matriz de características correspondiente a cada posición posible del gato y del ratón en el set de datos `X` que se entrega en el código.
Concretamente, tu tarea consiste en desarrollar un vector `y` que contenga la acción a seguir por el agente para cada estado del mundo posible, almacenados en la matriz `X`:

- Se tiene una matriz de datos `X` de dimensiones: `(free_positions`^{2}`, 4)`, donde el 4 representa un vector con las posiciones del gato y del ratón `(cat_x, cat_y, mouse_x, mouse_y)` y `free_positions`^{2} representa la cantidad de posiciones posibles en el mapa elevado al cuadrado.
- Además deberás crear un "vector"de etiquetas `y`, dentro del cual se debe asignar un movimiento a cada fila de la matriz `X` entregada. La forma de asignar el movimiento es mediante la representación one-hot de la acción electa por el agente, por ejemplo, la acción 3 (moverse hacia la derecha) será representada según el vector `(0, 0, 0, 1, 0)`, recordando que tanto las acciones como los índices del vector comienzan desde el número cero.

Acción	One-Hot				
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	0	1

Cuadro 2: Representación en One-Hot para cada acción del agente

  Debido a que cada entrada del vector `y` debe tener ese formato, este realmente se tratará de una matriz de tamaño `(free_positions`^{2}`, 5)`, es decir, contiene la representación en one-hot de la acción escogida para cada par de posiciones posible entre gato y ratón.

La forma exacta en la que se elige que movimiento realizar para cada fila de la matriz **X** queda a tu criterio. Este proceso es idéntico para el modelo del gato y para el del ratón. Lógicamente, la diferencia estará en la forma en la que cada uno decide que movimiento hacer dado el estado del mundo.

Recuerda que dispones de toda la información sobre el entorno y políticas de comportamiento de base para generar el vector `y`.

Actividad 1.2: diseño y creación de la Red Neuronal

En esta parte deberás crear el modelo de la Red Neuronal que posteriormente entrenarás. Para hacer esto deberás utilizar las librerías Scikit Learn o Keras.

Dentro del archivo `train_neural_agent.ipynb` encontrarás todo lo necesario para poder implementar y compilar una red básica utilizando Keras.

Actividad 1.3: entrenamiento del modelo

Finalmente, el útlimo paso que deberás realizar en el archivo `train_neural_agent.ipynb` es entrenar el modelo sobre el set de datos generado. Si bien se trata de una tarea sencilla, puede tomar algo tiempo, además, deberás ajustar los parámetros de tu modelo a modo de maximizar el *accuracy* sobre el set de entrenamiento (esto se conoce como *overfitting*, discutiremos sobre ello más adelante).

Una vez entrenado, deberás guardar el modelo en la carpeta `agents/data/` para poder ser utilizado por el agente.

Actividad 1.4: comportamiento del agente basado en redes neuronales

Una vez entrenada la red que modela el comportamiento del agente, deberás integrar su funcionamiento dentro del archivo **`agents/neural.py`** para obtener el siguiente movimiento por parte del agente.

En esta parte solo debes modificar las lineas indicadas en el método **get_action**, para obtener el movimiento y luego retornarlo.

Actividad 1.5: análisis

El *overfitting* es un efecto que normalmente intentamos evitar en las Redes Neuronales. Sin embargo, en esta tarea en particular, si deseamos que la red maximize su *accuracy* en el set de entrenamiento, lo que normalmente significa "aprenderse"de memoria los datos.

- Explique por qué para esta implementación en particular es conveniente tener *overfitting* sobre los datos ¿es este el caso para la mayoría de las aplicaciones?.
- Indica dos contraejemplos en donde no deseemos tener *overfitting* sobre nuestros datos. (Pista: investiga sobre el concepto de generalización)

Con respecto a las Redes Neuronales que decidiste implementar, responde las siguientes preguntas:

- ¿Por qué se utiliza la función de activación *softmax* en la salida de la red? Explica brevemente
- ¿Cuántas capas contiene tu modelo? ¿Cuántas neuronas contiene cada capa? Se espera que expliques tu razonamiento y/o proceso para llegar a esos números.

Parte 2: implementación de Aprendizaje Reforzado (2,5 ptos.)

Basándote en lo visto en clases y ayudantías, implementa el algoritmo *Q-Learning*[1] para ambos agentes del juego, es decir, el gato y el ratón. Para esto deberás realizar los siguientes pasos:

Actividad 2.1: clase ReinforcedAgent

1. Implementa el método `get_action`, que le permite al agente escoger un movimiento a realizar a partir de los valores de Q-Table.
2. Implementa el método `update_policy`, donde se actualizan los valores asociados a un par estado-acción de la Q_table a partir de su estado anterior y nuevo estado.

Actividad 2.2: clase RLCat

Esta case se encuentra encargada de modela el comportamiento del gato dentro del juego.

1. Implementa el método `get_reward` para otorgar una recompensa al gato a partir de una acción tomada, es decir, crea una recompensa que incentive la captura del ratón.
2. En el método `update_exploration` se debe implementar la actualización del exploration rate, utilizando los parametros de exploración asociados al gato.

Actividad 2.3: clase RLMouse

Esta case se encuentra encargada de modela el comportamiento del ratón dentro del juego.

1. Implementa el método `get_reward` para otorgar una recompensa al ratón a partir de una acción tomada, es decir, crea una recompensa que le incentive a escapar del gato.
2. En el método `update_exploration` se debe implementar la actualización del exploration rate, utilizando los parametros de exploración asociados al gato.

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

Actividad 2.4: implementación de Q-Learning

Una vez hayas programado los métodos de cada agente y estos sean capaces de aprender, entrénalos utilizando distintos hiper-parámetros y responde las siguientes preguntas:

- ¿Qué rol cumple la tasa de descuento en *Q-Learning*?
- ¿Qué tasa de descuento te dio mejores resultados? ¿Por qué crees que fue así?
- ¿Para qué sirve el *learning rate*? ¿Qué valor te entregó mejores resultados? Comenta al respecto.
- Explica cómo diseñaste el reward de cada agente y la lógica detrás de tu decisión. Piensa en otro reward alternativo para cada agente (no la implementes), entrega una ventaja y una desventaja en comparación con el reward implementado.

Parte 3: analisis comparativo (1 pto.)

En esta sección, se pide realizar un análisis comparativo entre los dos modelos que has implementado: el agente controlado por la Red Neuronal y el agente controlado por el Aprendizaje Reforzado. Deberás hacer pruebas y analizar tus resultados para responder las siguientes preguntas:

- ¿Cuál de los dos modelos es más eficiente en términos de tiempo de entrenamiento? ¿Por qué crees que es así?
- ¿Cuál de los dos modelos tuvo un mejor desempeño en el juego? ¿Qué métricas utilizas para cuantificar su desempeño?
- ¿Qué sucede si el tamaño del mapa cambia? ¿Cuál de los dos modelos maneja mejor este cambio y por qué?
- Menciona dos diferencias observables en el comportamiento de cada agente, por comportamiento nos referimos a secuencias de movimientos o situaciones que frecuenta el agente.

Por último, reflexiona y explica cuál de los dos modelos recomendarías para desarrollar agentes en juegos de caza más complejos y por qué.

Bonus: DCCampeonato (0,5 ptos.)

¿Tienes al mejor agente del curso? Es hora de demostrarlo. En los próximos días lanzaremos el DCCampeonato, una competencia que desafiará tu habilidad para desarrollar el modelo superior.

Cada agente competirá en enfrentamientos emocionantes y desafiantes, y el ganador no solo ganará la gloria del reconocimiento de sus compañeros, sino que también será recompensado con 5 décimas adicionales en su tarea.

Las bases del torneo serán anunciadas pronto. ¡Estudia, investiga y perfecciona tu modelo, prepárate para competir!