CI-0148 Aprendizaje Máquina
Prof. Pablo Sauma Chacón

Laboratorio 8

Objetivo: Familiarizar al estudiante con la creación y el uso del algoritmo de aprendizaje por refuerzo Deep Q-Learning en un entorno controlado.

Enunciado: Lea las instrucciones del documento y resuelva el enunciado en Python. Al inicio de su documento adjunte su nombre y carnet como un comentario. Este trabajo es de carácter individual, tampoco se aceptarán códigos que no hayan sido desarrollados por su persona.

Para este laboratorio se trabajará con el juego clásico de Snake. Con este fin su profesor implementó un ambiente de pruebas y una interfaz gráfica, que puede descargar aquí.

El código permite entrenar a un agente de aprendizaje por refuerzo que interactúa con el juego por medio de los llamados `get_state()`, `is_terminal_state()` y `perform_action()`. El objetivo es entrenar a su agente para que logre maximizar su puntaje/comer la mayor cantidad de manzanas posibles antes de perder.

Dentro del código encontrará la clase `Agent` con tres funciones que son las que debe implementar:

1. Método de inicialización `__init__(self, memory_capacity, batch_size, c_iters, learning_rate, discount_factor, eps_greedy, decay)` que inicializa el agente que utiliza Deep Q-Learning. Dada la complejidad del problema se le recomienda utilizar un modelo de dos redes neuronales: la red `target` y la red `policy` (pero si no lo desea no es obligatorio hacerlo así). El parámetro `memory_capacity` especifica el tamaño de la memoria (dado que las simulaciones son relativamente pequeñas, puede ser útil guardar "ejemplos" como parte del entrenamiento, el tamaño lo que limita es la capacidad máxima) El parámetro `batch_size` especifica el tamaño de la muestra que se tomará de la memoria para cada entrenamiento. El parámetro `c_iters` especifica la cantidad de sesiones de entrenamiento antes de que se actualicen los pesos de la red target con los de la red policy. Los parámetros `learning_rate`, `discount_factor`, `eps_greedy` y `decay` que son parámetros propios del algoritmo. Para fines de la implementación se asumirá que el `decay` aplica exclusivamente al valor de epsilon-greedy y la tasa de aprendizaje se utilizará para la inicialización del optimizador.
    a. Considere que puede resultarle de utilidad contar con dos redes neuronales, la de `target` y la de `policy`. Actualizando los pesos de la de `target` cada `c_iters`.
    b. Recuerde que al generar valores con la red `target`, dado que estos valores no serán usados como parte de un entrenamiento (se utilizan solo para calcular el error de `policy`), entonces no es necesario calcular los gradientes, por lo que el

comando `torch.no_grad()` y `.detach()` le ayudarán a reducir la complejidad de su programa.

c. Recuerde que el objetivo del entrenamiento es aproximar los valores Q(s) predichos por su red `policy` a los valores reales asociados dados por la recompensa obtenida junto al valor $\gamma \cdot Q(s')$.

2. Método de simulación `simulation(self, env)` que ejecuta una simulación del agente. El parámetro `env` es la variable de tipo `Snake` que corresponde al entorno de nuestro agente. Recuerde que para ejecutar una simulación debe reiniciar el entorno, y luego seguir ejecutando acciones (en este caso un llamado al método `step` del punto 3) hasta que se haya alcanzado un estado terminal. Note que el objeto `env` posee llamados para reiniciar el entorno y para preguntar si el estado actual es un estado final. Al final de cada simulación es necesario reajustar el valor epsilon-greedy acorde a la tasa de decaimiento.

3. Método de ejecución `step(self, env, learn=True)` que ejecuta un paso del agente. El parámetro `env` es la variable de tipo `Snake` que corresponde al entorno de nuestro agente. El parámetro `learn` indica si el agente está ejecutando en modo aprendizaje o no. De estar ejecutando en modo aprendizaje, el agente deberá generar una variable aleatoria y comparar su valor con el epsilon-greedy actual. De ser menor deberá elegir una acción al azar (modo exploración), de ser mayor (o si `learn=False`) elegirá la mejor acción conocida hasta ahora. Note que el objeto `env` posee un método para obtener el estado actual, además de un método para ejecutar una acción, la cuál retorna tanto la recompensa obtenida como el nuevo estado alcanzado. De estar ejecutando en modo aprendizaje deberá agregar la tupla (estado, acción, nuevo-estado, recompensa) a la memoria del agente.

4. Finalmente, pruebe su agente. Note que como parte de este laboratorio no solo se le permite modificar los parámetros de entrenamiento de su agente, sino también los valores de las recompensas generadas por diferentes eventos y también la manera en la que se retorna el estado: actualmente el estado está codificado como una imagen multi-capa, pero si lo desea, puede trabajar con otra codificación de la información del estado como entrada a su red neuronal.
    a. Documente los cambios realizados y su solución encontrada.
    b. Documente el mejor resultado obtenido para su modelo, la cantidad de iteraciones y los hiperparámetros seleccionados.