PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
IIC-3697 APRENDIZAJE PROFUNDO

TAREA 3: APRENDIZAJE REFORZADO UTILIZANDO TÉCNICAS DE APRENDIZAJE PROFUNDO

Fecha Máxima de Entrega: Miércoles 27 de Junio

Estimad@s, en esta tarea tendrán la oportunidad de experimentar con el uso de técnicas de aprendizaje profundo para el entrenamiento de un agente que aprende una política de acción en base a refuerzos. En particular, implementarán el algoritmo DQN [1] que vimos en clases. Tal como discutimos, entre su amplio abanico de aplicaciones, este algoritmo permite aprender políticas de acción para juegos de Atari. Entre estos, en esta tarea nos centraremos en el juego Pong.

Para el desarrollo de esta tarea nuevamente utilizarán la plataforma Google Colaboratory. En términos de herramientas de software podrán usar Keras, TensorFlow o Pytorch. Adicionalmente, utilizarán OpenAI Gym (`http://gym.openai.com/`, un toolkit que facilita la experimentación con algoritmos de aprendizaje reforzado. De este toolkit usaremos su implementación del juego Pong-V0. Para instalar OpenAI Gym y sus librerías de Atari puede utilizar **pip** mediante los siguiente 2 comandos:

```
pip install gym
pip install gym[atari]
```

Pong-V0

La figura 1 muestra la interface del juego Pong proporcionada por OpenAI Gym, para más detalle ver `http://gym.openai.com/envs/Pong-v0/`.

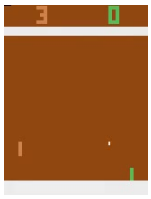

Figura 1: Interface de juego Pong-v0 provista por el toolkit OpenAI Gym.

1.1 Estados

En términos del estado del juego, la interface de OpenAI Gym retorna imágenes de 210x160x3, donde 210x160 indica la resolución espacial de cada cuadro, y 3 indica las componentes de color RGB. Cada pixel es representado por valores en el rango [0,255] (uint8). Tal como en DQN, para agilizar el aprendizaje en su tarea desarrollarán código para bajar la resolución de cada cuadro a 80x80x1, es decir, operarán sobre imágenes en niveles de gris con un tamaño de 80x80 pixels. Para esto pueden usar diversas librerías de Python, tales como Skimage, PIL, y OpenCV-Python, entre otras. A modo de ejemplo, el siguiente código muestra como realizar esto utilizando Skimage:

```
%Convertir imagen a scala de gris y re-escalar
Img = skimage.color.rgb2gray(inImage)
Img = skimage.transform.resize(Img,(80,80))
```

```
%Concatenar 4 frames
Im4Frames = np.stack((Img, Img, Img, Img), axis=2)
```

Adicionalmente, también reduciremos la resolución temporal del juego. Para esto, al seleccionar una acción repetiremos su ejecución durante 4 instantes consecutivos de tiempo. Esto permitirá adquirir 4 observaciones consecutivas de la interface del juego. Luego, aplicaremos sobre estos 4 cuadros un operador de max-pooling en la dimensión temporal, guardando para cada posición espacial el valor máximo de cada pixel. De esta manera, la imagen resultante del juego tendrá un tamaño de 80x80x1, la cual corresponderá a la observación utilizada para entrenar el algoritmo DQN.

1.2 Acciones

En términos de acciones, en el caso del juego Pong hay 3 posibles acciones: mover arriba, mover abajo, no mover. Sin embargo, la interface de OpenAI Gym considera 6 posibles acciones, de las cuales sólo 3 son de utilidad.

1.3 Refuerzos

En términos de refuerzo (reward), en el juego Pong gana el jugador que primero alcance 21 puntos. En la interface de OpenAI Gym, durante la ejecución de cada punto el reward es 0, mientras que al finalizar el punto el ganador recibe un reward de +1 y el perdedor recibe un reward de -1. Cada episodio considera un partido completo, es decir, termina cuando uno de los 2 jugadores alcanza 21 puntos. De esta manera, en cada episodio un jugador recibe un reward final que está en el rango [-21,21].

La interface de OpenAI Gym incorpora un jugador pre-entrenado, el cual tiene un nivel de destreza capaz de vencer a un humano promedio. De esta manera, ustedes tendrán que vengar a nuestra especie programando un jugador que pueda vencer al jugador pre-programado de OpenAI Gym. Es decir, si todo marcha bien, obtendrán un jugador con capacidades super humanas, aunque probablemente esto implique que ustedes tampoco lo podrán vencer :). Como dice el dicho de un buen alumno de AI del DCC: *"Si no los puedes vencer, programa un agente que si pueda :)"*.

Como mencionamos en clases, para el caso de espacios continuos de estado, como es el caso del juego Pong, no es posible mantener una tabla de la función $Q(s, a)$ para todas las combinaciones estado-acción. En forma alternativa, en nuestro caso usaremos una red convolucional para estimar el valor de la función $Q(s, a)$.

DQN

Para implementar DQN tendrán que resolver las ecuaciones que estiman la función $Q(s, a)$, es decir:

$$Q(s, a) = r(s, a) + \gamma V^*(s', a')$$
$$Q(s, a) = \mathbb{E}_{s' \sim env}[r(s, a) + \gamma \operatorname*{argmax}_{a'} Q(s', a')]$$

Como vimos en clases, la aproximación directa de la función $Q(s, a)$ mediante una red neuronal se ve afectada por inestabilidad en el aprendizaje, lo cual no permite la convergencia a una política adecuada de acción. Entre otras cosas, esto se debe a que los pares (s, a) explorados durante un mismo episodio están fuertemente correlacionados, lo cual rompe el supuesto de que los ejemplos de entrenamiento son independientes e identicamente distribuidos (iid), condición necesaria para garantizar la convergencia de métodos de optimización basados en descenso de gradiente. Afortunadamente, DQN nos permitirá superar esta limitación.

Específicamente, la red utilizada por DQN tendrá como entrada el estado del juego, representado por una observación correspondiente al resultado de la operación de max-pooling sobre los últimos 4 cuadros del juego, mencionado anteriormente. A su vez, como salida, la red entregará el valor $Q(s, a)$ asociado a cada posible acción a ejecutar. De esta manera, la estimación de la función Q considera para cada entrada s la estimación de $Q(s, a_i)$, con $i \in [0 \ldots, 5]$.

Para implementar el algoritmo DQN, su código debe incorporar las 3 estrategias utilizadas por este algoritmo para converger a una estimación adecuada de la función $Q(s, a)$, según el siguiente lineamiento:

1. Uso de un buffer de memoria para almacenar la experiencia adquirida durante la ejecución del juego (Experience Replay). Tal como DQN, utilicen un buffer de 1M de pasos (steps). Adicionalmente, siguiendo la implementación de DQN y tal como el código que vimos en clases para el juego Flappy Bird, durante los primero 50.000 steps seleccionen acciones bajo una política uniforme y sin realizar entrenamiento, sólo llenar el buffer de memoria.
2. Uso de una red neuronal independiente para estimar la función $\hat{Q}(s, a)$, la cual es considerada como función objetivo durante el entrenamiento, es decir, proporciona los rótulos (labels). Tal como en DQN, la red que estima $\hat{Q}(s, a)$ es actualizada cada C pasos utilizando el valor de los pesos de la red que estima $Q(s, a)$.
3. Estrategia de exploración tipo ϵ-greedy. A modo de ejemplo, la estrategia considerada por DQN es decrementar ϵ desde 1 a 0.1 en forma lineal durante el primer millón de pasos (steps), y luego mantener el valor constante en 0.1.

Para la red convolucional utilice la misma arquitectura usada por DQN, incluyendo la entrada correspondiente a las últimas 4 observaciones del juego (obs: no confundir con la aplicación del operador max-pooling mencionado anteriormente, esto se refiere a observaciones finales después de la aplicación de reducción temporal con el operador max-pooling).

Actividad 1

¿Cuál es el número de parámetros utilizado por el modelo?.

Actividad 2

Realice un gráfico que muestre la evolución del aprendizaje a través del entrenamiento. Específicamente, construya un gráfico que muestre el valor del score promedio por episodio en función de las épocas de entrenamiento. Para construir este gráfico considere que una época está dada por el procesamiento de 250 mil ejemplos (pasos) de entrenamiento. Para el cálculo del score promedio considere 100 episodios.

Hint: Como referencia, el score de un humano promedio jugando contra el agente pre-programado de OpenAI Gym es de alrededor de -3. Al comienzo del entrenamiento, su agente entrenado con DQN debería tener scores cercanos a -21 (por qué?), para luego alcanzar scores cercanos a 0 después de 4-6 épocas, y scores positivos (> 5 o incluso > 10) después de unas 10 a 20 épocas.

Actividad 3

Indique todos los parámetros relevantes utilizados para su implementación, tales como tasa de aprendizaje, método de optimización, tamaño de batch, etc.

En su informe de tarea debe contestar las 3 actividades propuestas e incluir su código. Recuerde que, si bien puede discutir con sus compañeros acerca de la tarea, la implementación final debe ser realizada en forma exclusiva por los miembros de su grupo. Cualquier duplicidad de código será considera como caso de copia. Esto incluye códigos descargados desde Internet.

Si bien el código para implementar DQN es relativamente simple, no más de 50 líneas, el entrenamiento es bastante intenso. Para poder lograr un buen jugador de Pong-v0 el proceso de entrenamiento debería ser de alrededor de 12 horas, así que no deje el desarrollo de la tarea para último momento. Para los que quieran utilizar su CPU en lugar de las GPU de Google Colaboratory, el tiempo de entrenamiento estará en el rango 24 a 48 horas. Suerte con todo!.

Bibliografía

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. Kingand D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.