Introducción del algoritmo de aprendizaje por refuerzo DQN para juegos de Atari

Angel Larreategui Castro
Universidad Nacional de Ingeniería
Lima, Perú
alarreateguic@uni.pe

Fernando Zambrano Altamirano
Universidad Nacional de Ingeniería
Lima, Perú
fzambranoa@uni.pe

José Reyes Gutiérrez
Universidad Nacional de Ingeniería
Lima, Perú
jreyesg@uni.pe

***Resumen*—En el presente trabajo se llegará a mostrar el modelo de aprendizaje profundo o conocido por sus siglas como DQN. El presente modelo es una red neuronal convolucional, la cual va a ser entrenada con una variante de aprendizaje Q, en donde el input o entrada serán píxeles sin procesar y el output será una función que va estimar futuras recompensas. En nuestro trabajo se pasará a aplicar lo mencionado a un juego del Atari 2600, una consola de video juegos muy exitosa y polémica en su momento, que se pasará a ver en el presente trabajo.**

Índice de Términos— Deep Learning, DQL, Atari, Atari 2600, PyTorch, TensorFlow

I. Introducción

Ya que los videojuegos son un desafío de varias tareas y de toma rápida de decisiones, se busca que un algoritmo pueda adaptarse a esto con miras a poder adaptarse a otras multitareas con la misma eficiencia.

I-A. Objetivo del Estudio

Analizar el desempeño de la IA y ver que alcance el mayor rendimiento posible en juegos de la consola Atari 2600 (Constantemente superándose). Esto debido a que si una IA logra optimizar su rendimiento en actividades que requieren varias acciones al mismo tiempo (Como en un videojuego) luego se podrá aplicar dicho entrenamiento en otros campos de manera análoga.

II. Marco Teórico

II-A. Deep Learning

Es un conjunto de algoritmos de aprendizaje automático que intenta modelar abstracciones de alto nivel en datos usando arquitecturas computacionales que admiten transformaciones no lineales múltiples e iterativas de datos expresados en forma matricial o tensorial. El aprendizaje profundo es parte de un conjunto más amplio de métodos de aprendizaje automático basados en asimilar representaciones de datos. Una observación (por ejemplo, una imagen) puede ser representada en algunas formas (por ejemplo, un vector de píxeles), pero algunas representaciones hacen más fácil aprender tareas de interés (por ejemplo, "¿es esta imagen una cara humana?") sobre la base de ejemplos, y la investigación en esta área intenta definir qué representaciones son mejores y cómo crear modelos para reconocer estas representaciones.

II-B. Q-Learning

En el problema completo de aprendizaje por refuerzo, el estado cambia cada vez que ejecutamos una acción. Podemos representar el problema de la siguiente manera. El agente recibe el estado (**state**) en el que se encuentra el entorno (**environment**), el cual representaremos con la letra s (**state**). El agente ejecuta entonces la acción que elija, representada con la letra a (**action**). Al ejecutar esa acción, el entorno responde proporcionando una recompensa, representada con la letra r (**reward**), y el entorno se traslada a un nuevo estado, representado con s' (**next state**). Este ciclo se puede observar en la figura 1.

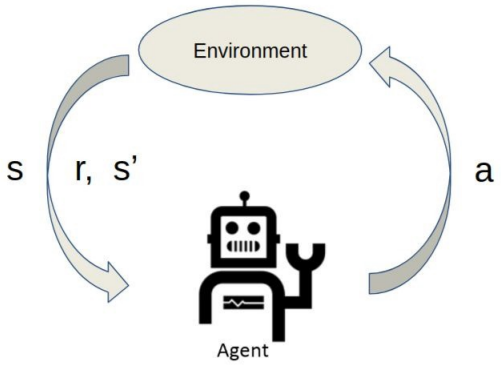

Figura 1. Representación del cambio de estado cada vez que se ejecuta una acción [1]

Por lo tanto, la acción que el agente escoja no debe sólo depender de la recompensa a que vaya a recibir a corto plazo. Debe elegir las acciones que a largo plazo le traerán la máxima recompensa (o retorno) posible en todo el episodio (**episode**). Este ciclo trae una secuencia de estados, acciones y recompensas, desde el primer paso del ciclo hasta el último: $s1, a1, r1; s2, a2, r2; \ldots; sT, aT, rT$. Aquí, T indica el fin del episodio.

II-B1. Función de valor: Para cuantificar cuanta recompensa obtendrá el agente a largo plazo desde cada estado, introducimos la función de valor $V(s)$. Esta función produce una estimación de la recompensa que obtendrá el agente hasta el final del episodio, empezando desde el estado s. Si conseguimos estimar este valor correctamente, podremos decidir ejecutar la acción que nos lleve al estado con el valor más alto.

II-B2. Q-Learning, resolviendo el problema: Para resolver el problema del aprendizaje por refuerzo, el agente debe aprender a escoger la mejor acción posible para cada uno de los estados posibles. Para ello, el algoritmo Q-Learning intenta aprender cuanta recompensa obtendrá a largo plazo para cada pareja de estados y acciones (s, a). A esa función la llamamos la función de acción-valor (action-value function) y este algoritmo la representa como la función $Q(s, a)$, la cual devuelve la recompensa que el agente recibirá al ejecutar la acción a desde el estado s, y asumiendo que seguirá la misma política dictada por la función Q hasta el final del episodio. Por lo tanto, si desde el estado s, tenemos dos acciones disponibles, $a1$ y $a2$, la función Q nos proporcionará los valores-Q (Q-values) de cada una de las acciones. Por ejemplo, si $Q(s, a1) = 1$ y $Q(s, a2) = 4$, el agente sabe que la acción $a2$ es mejor y le traerá mayor recompensa, por lo que será la acción que ejecutará.

II-B3. Ecuación de Bellman: Primero veamos como es la ecuación de Bellman:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

Figura 2. la ecuación de Bellman [1]

La explicación para esta ecuación es la siguiente. El valor-Q del estado s y la acción a $(Q(s, a))$ debe ser igual a la recompensa r obtenida al ejecutar esa acción, más el valor-Q de ejecutar la mejor acción posible a' desde el próximo estado s', multiplicado por un factor de descuento γ (discount factor), que es un valor con rango $\gamma \in (0, 1]$. Este valor γ se usa para decidir cuánto peso le queremos dar a las recompensas a corto y a largo plazo, y es un hiperparámetro que debemos decidir nosotros.

II-C. El algoritmo Deep Q-Network o DQN

Vimos que el algoritmo Q-Learning funciona muy bien cuando el entorno es simple y la función $Q(s, a)$ se puede representar como una tabla o matriz de valores. Pero cuando hay miles de millones de estados diferentes y cientos de acciones distintas, la tabla se vuelve enorme, y no es viable su utilización. Por ello, Mnih et al. [1] inventaron el algoritmo Deep Q-Network o DQN. Este algoritmo combina el algoritmo Q-learning con redes neuronales profundas (Deep Neural Networks). Como es sabido en el campo de la IA, las redes neuronales son una fantástica manera de aproximar funciones no lineales. Por lo tanto, este algoritmo usa una red neuronal para aproximar la función Q, evitando así utilizar una tabla para representar la misma. En realidad, utiliza dos redes neuronales para estabilizar el proceso de aprendizaje. La primera, la red neuronal principal (main Neural Network), representada por los parámetros θ, se utiliza para estimar los valores-Q del estado s y acción a actuales: $Q(s, a; \theta)$. La segunda, la red neuronal objetivo (target Neural Network), parametrizada por θ', tendrá la misma arquitectura que la red principal pero se usará para aproximar los valores-Q del siguiente estado s' y la siguiente acción a'. El aprendizaje ocurre en la red principal y no en la objetivo. La red objetivo se congela (sus parámetros no se cambian) durante varias iteraciones (normalmente alrededor de 10000), y después los parámetros de la red principal se copian a la red objetivo, transmitiendo así el aprendizaje de una a otra, haciendo que las estimaciones calculadas por la red objetivo sean más precisas.

III. Estado del Arte

III-A. Playing Atari with Deep Reinforcement Learning [2]

Un gran avance para los proyectos de este tipo sería el modelo de control de políticas basado en deep learning el cual recibe una entrada de pixeles sin procesar y da como salida una estimación de recompensas futuras. Un modelo así es de mucha ayuda para el entrenamiento del algoritmo propuesto.

III-B. MODEL BASED REINFORCEMENT LEARNING FOR ATARI [3]

Otro modelo exitoso que también sirve de referencia es el Simulated Policy Learning (SimPLe) (Kaiser et al., 2020), el cual está basado en modelos de predicción de video y presenta una comparación de muchas arquitecturas de modelo, incluyendo una nueva arquitectura que da los mejores resultados en su configuración.

III-C. State of the Art Control of Atari Games Using Shallow Reinforcement Learning [4]

Por último, cabe mencionar el estudio que usa aprendizaje por refuerzo poco profundo para juegos de Atari (Liang et al., 2016), el cual plantea un conjunto de características computacionalmente prácticas que logran un rendimiento capaz de competir con el de algoritmos DQN en el Arcade Learning Environment (ALE).

IV. Metodología

En esta sección se describen las herramientas y la métodología que se usarán en el presente trabajo

IV-A. Pytorch

Es una biblioteca de tensores optimizada para el aprendizaje profundo mediante GPU y CPU. [5]

Es una biblioteca de aprendizaje automático de código abierto basada en la biblioteca de Torch, utilizado para aplicaciones que implementan cosas como visión artificial y procesamiento de lenguajes naturales, principalmente desarrollado por el Laboratorio de Investigación de Inteligencia Artificial de Facebook (FAIR) Es un software libre y de código abierto liberado bajo la Licencia Modificada de BSD. A pesar de que la interfaz de Python está más pulida y es el foco principal del desarrollo, PyTorch también tiene una interfaz en C++. [6]

IV-B. TensorFlow

Es una biblioteca de código abierto tanto para Machine Learning como para la computación numérica a gran escala. También, esta biblioteca reúne una serie de modelos y algoritmos de Deep Learning y Machine Learning y los hace útiles mediante una metáfora común. [7]

TensorFlow puede entrenar y ejecutar redes neuronales profundas para la clasificación de dígitos escritos a mano, el reconocimiento de imágenes, la incrustación de palabras, las redes neuronales recurrentes, los modelos secuencia a secuencia para la traducción automática, el procesamiento del lenguaje natural y las simulaciones basadas en ecuaciones diferencias parciales. Además, TensorFlow admite predicción de producción a escala, con los mismos modelos utilizados para el entrenamiento.

IV-C. Metodología de trabajo

El método a seguir para implementación básica de un algoritmo por refuerzo DQN para juegos de Atari es el siguiente

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figura 3. Aprendizaje DQN con repetición de experiencia.

V. Experimentación y resultados

La experimentación se ejecutó en Google Colab en un entorno con GPU habilitado. Algo que se ha notado mientras se ejecutaba el código ha sido que ha habido una demora notable y eso se ha debido, a nuestro parecer, a que ha estado entrenando el juego y hemos obtenido los siguientes gráficos.

Algorithm 1: Count-based exploration through static hashing, using SimHash

Define state preprocessor $g : \mathcal{S} \to \mathbb{R}^D$
(In case of SimHash) Initialize $A \in \mathbb{R}^{k \times D}$ with entries drawn i.i.d. from the standard Gaussian distribution $\mathcal{N}(0, 1)$
Initialize a hash table with values $n(\cdot) \equiv 0$
for each iteration j **do**
  Collect a set of state-action samples $\{(s_m, a_m)\}_{m=0}^M$ with policy π
  Compute hash codes through any LSH method, e.g., for SimHash, $\phi(s_m) = \text{sgn}(Ag(s_m))$
  Update the hash table counts $\forall m : 0 \le m \le M$ as $n(\phi(s_m)) \leftarrow n(\phi(s_m)) + 1$
  Update the policy π using rewards $\left\{ r(s_m, a_m) + \frac{\beta}{\sqrt{n(\phi(s_m))}} \right\}_{m=0}^M$ with any RL algorithm

Figura 4. Exploración count-based a través de static hashing usando SimHash.

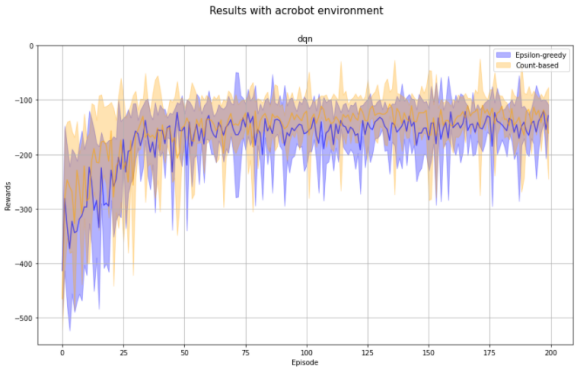

Figura 5. Resultados con el entorno de Acrobot.

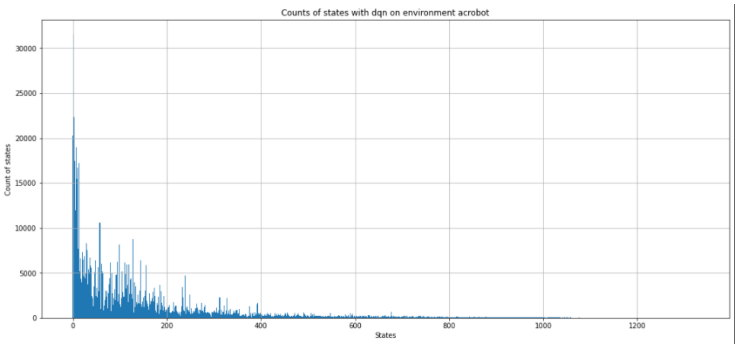

Figura 6. Conteo de estados con el DQN del entorno del Acrobot.

VI. Discusión de resultados

Tal y como se mencionó en el apartado anterior, algo a notar ha sido el enorme tiempo de espera que hemos observado al compilar determinadas celdas. Esto obviamente se va a notar cuando se vean ejecutadas las celdas en nuestro Google Collab que se adjuntó con los demás enlaces para el tercer entregable del presente proyecto. Algo también muy notable ha sido la mejora del puntaje en el juego Acrobot, y esto como se explicó se ha debido gracias a que mientras iba adquiriendo más conocimientos se ha obtenido una mejora sustancial.

VII. Conclusiones

Los avances en IA si bien a corto plazo parecen pequeños, las repercusiones de estos son realmente importantes en este campo. En nuestro caso particular, muchos no verían utilidad en que una IA sea entrenada en videojuegos y logre optimizar su puntaje. Pero el valor de esto radica en lograr que la IA aprenda a realizar varias tareas que requieren decisiones rápidas de modo óptimo, y eso se puede extrapolar a otros campos que, análogamente, requieren realizar varias tareas a la vez y tomar decisiones rápidamente. Los resultados obtenidos de trabajos como el nuestro parecerán un pequeño paso, pero son en realidad un gran salto para el campo. En un futuro podríamos ver aplicaciones tales como robots que realizan tareas tales como medicina, transporte, entre otras, y que actúan óptimamente gracias a un entrenamiento como el presentado en este trabajo.

VIII. Trabajos Futuros

En el presente trabajo hemos visto como a medida que se ha ido entrenando el juego Acrobot de Atari mediante el algoritmo de aprendizaje por refuerzo DQN se ha ido mejorando el puntaje del mismo de una manera impresionante, ¿por qué ocurre esta mejora?, la respuesta parece más sencilla de lo que parece e inclusive es muy evidente. Al inicio al ir entrenando empieza a jugar al juego sin saber mucho de lo que exactamente va a hacer, sin embargo, a medida que juega una y otra vez se va aprendiendo mejor que es lo que se va a hacer, y por ende, hay esa mejora sustancial en el puntaje del mismo.

Así como se realizó e implementó este algoritmo de aprendizaje para juegos de Atari, en trabajos futuros y gracias a que la tecnología va en constante avance, se podría implementar dicho algoritmo no solo para juegos de Atari, sino también para más consolas retro tales como la Sega Génesis, Nintendo 64, NES, PSX, y así sucesivamente hasta inclusive poder llegar a las consolas de actual generación. Sin lugar a dudas sería muy interesante ver todo lo que se acaba de mencionar, y se espera verlo en trabajos futuros.

Referencias

[1] Introduccion al aprendizaje por refuerzo q-learning Disponible en https://markelsanz14.medium.com/introducción-al-aprendizaje-por-refuerzo-parte-2-q-learning-883cd42fb48e.

[2] Playing Atari with Deep Reinforcement Learning Disponible en https://arxiv.org/pdf/1312.5602.pdf

[3] MODEL BASED REINFORCEMENT LEARNING FOR ATARI Disponible en https://arxiv.org/pdf/1903.00374.pdf

[4] State of the Art Control of Atari Games Using Shallow Reinforcement Learning Disponible en https://arxiv.org/pdf/1512.01563.pdf

[5] PYTORCH DOCUMENTATION Disponible en https://pytorch.org/docs/stable/index.html

[6] PyTorch Disponible en https://es.wikipedia.org/wiki/PyTorch

[7] ¿Qué es TensorFlow? ¿Cómo funciona? Disponible en https://aprendeia.com/que-es-tensorflow-como-funciona/