Computación Bioinspirada

PacMan con Aprendizaje profundo no supervisado.

Isabel Najarro Borrego

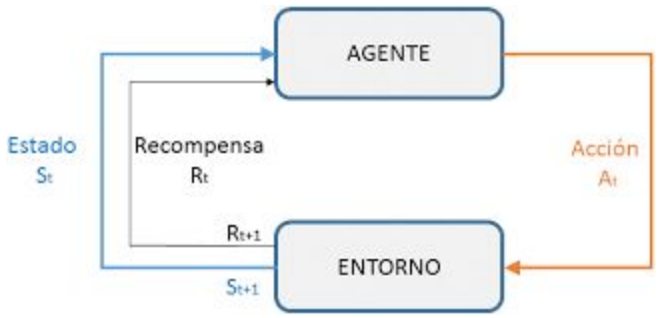
25 de enero del 2020

Índice

Introducción

El aprendizaje reforzado pertenece a un subcampo del aprendizaje automático en el que el agente de la máquina aprende el comportamiento óptimo en un ambiente de prueba y error. Este es puesto en un ambiente e interactúa con él y cualquier acción que realice puede ser recompensada positiva o negativamente. Con estas recompensas el agente aprende por sí mismo y sin datos del entrenamiento como en el aprendizaje supervisado.

El aprendizaje del agente se enmarca como un proceso de Decisión de Markov en el que el entorno está representado por el espacio de estado S y el agente P puede realizar cualquier acción de las acciones disponibles A y obtener una recompensa R. El objetivo consiste en maximizar esta recompensa por las acciones tomadas.

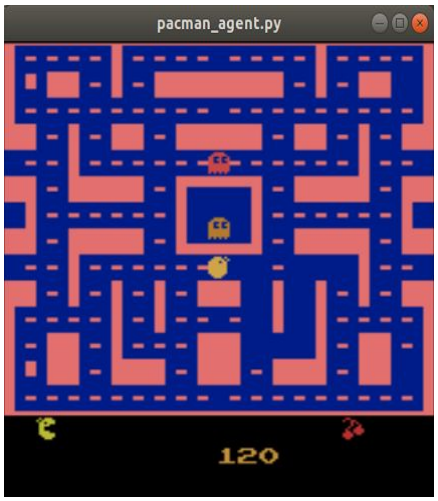


En el Q-Learning se calcula la función Q que devuelve la recompensa utilizada para proporcionar el refuerzo y se puede decir que representa la "calidad" para cada acción-estado. Por tanto, se usará un algoritmo de Q-Learning que utiliza redes neuronales profundas para aproximar la función Q. Estas redes ayudarán a extraer características significativas de los datos, reduciendo así el espacio de estado efectivo y también aprende los valores Q incorporando la recompensa obtenida en función de la pérdida.

Al ser un aprendizaje de refuerzo el agente no se contará con un conjunto de datos. En su lugar, nosotros se extraerán los datos a medida que avancemos en el entorno del juego.

Descripción del juego

Para llevar a cabo este juego se utilizará el entorno de PacMan de OpenAI Gym para recopilar datos y probar la solución. Cada partida consta de tres rondas en las que se tiene que conseguir la mayor puntuación posible.



PacMan es un juego de laberinto en el que un jugador tiene que comerse todos los puntos o pastillas del laberinto esquivando a los cuatro fantasmas que se encuentran en el mapa. Este mapa además cuenta con dos salidas en cada extremo para que PacMan pueda cambiar de cuadrante.

Los fantasmas salen de la caseta del medio y es la única zona a la que no puede acceder PacMan. Cada fantasma tiene un comportamiento distinto para atrapar al PacMan. Las rutas se definen tomando como referencia su propia posición y la de Pac-Man dentro del laberinto, aunque si hay mucha distancia los fantasmas no detectan a Pac-Man y se limitan a merodear por el laberinto hasta encontrarlo.

Cada cierto tiempo sale un objeto de Bonus (que se corresponde con la cereza) en el pasillo que está debajo de la caseta de los fantasmas para obtener más puntos, siendo cada vez mayor el número de puntos obtenidos según transcurren los niveles.

Además, habrá 4 pastillas en cada cuadrante del mapa y cuando las coma el Pacman podrá comerse a los fantasmas durante unos segundos. En estos segundos los fantasmas comidos vuelven a la caseta y tras pasar unos segundos vuelven al mapa.

La solución llevada a cabo consta de una red neuronal que utilizando el flujo tensorial y las líneas de base que toma la imagen del juego como entrada y produce la acción óptima para obtener la salida. Esta, se entrenará con el entorno del juego.

Sólo se entrenará el DQN en las recompensas positivas y los pesos de la red neural se ajustarán de manera que que el DQN pueda calcular una buena aproximación de la función Q óptima.

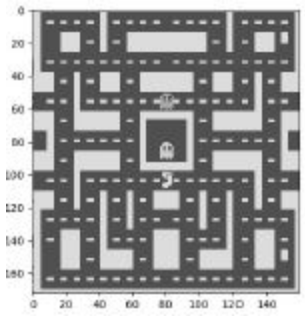
La muerte de PacMan no tendrá recompensa negativa.

Algoritmo

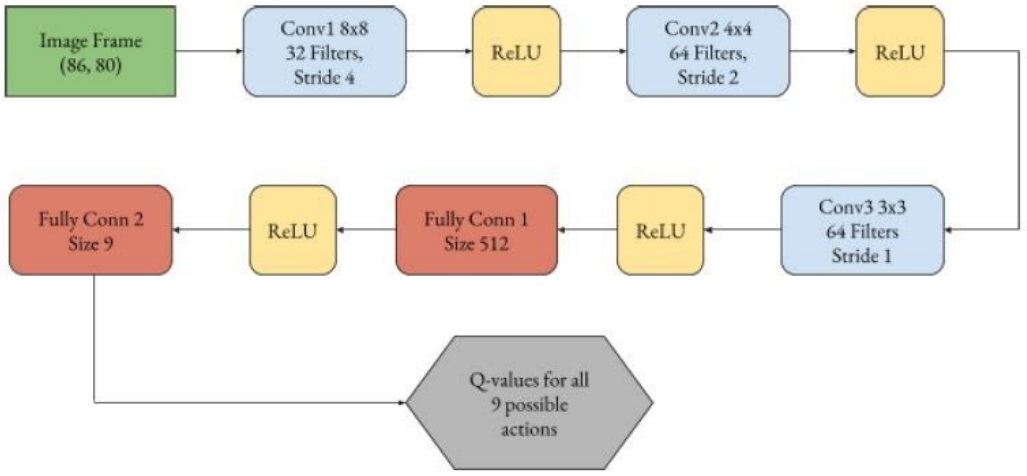
1. Inicializar la red neuronal con pesos aleatorios pequeños para calcular los valores Q.
2. Para cada step:
    - Obtener 4 fotogramas y el preproceso. Esto es el estado.
    - Decidir la acción que va a llevar a cabo PacMan utilizando el parámetro de exploración (Epsilon). Este se envía a la red para decidir la mejor acción a escoger.
    - Obtener la recompensa y el nuevo marco.
    - Almacenar en el Replay Buffer los resultados.
    - Usar la red para calcular los valores de Q para todas las acciones en el nuevo estado.
    - Utilizar la función de pérdida para calcular esta y retropropagar los gradientes a través de la red.
    - Repetir hasta fin de steps.

Implementación

Al tener la imagen del marco, tenemos toda la información que necesita la red neural para construir el estado. Al igual que en cualquier otro problema de RL (como el smartcab), los datos no están pregenerados, simulamos los datos en el momento de la formación.



Una vez que tenemos un marco que preprocesar llevaremos a cabo la conversión de la imagen a escala de grises, eliminación de la barra de estado en la parte inferior y escalar hacia abajo el tamaño a (86, 80) para reducir las dimensiones. La única información que se pierde es el número de vidas y el número de cerezas que nos quedan que son datos que no se van a utilizar para entrenar la red.



El modelo de la red está formado por tres capas convolucionales ReLU a las que les proceden dos capas densas ReLU las cuales se encargarán de dar una salida adaptada a la tabla Q con la que se trabaja.

Este marco se pasará a la red neuronal para que decida la acción óptima. Una vez que el agente decide la acción lo pasa al medio ambiente, la acción se ejecuta en el siguiente 2, 3 o 4 (el número de tramas está elegido al azar por el medio ambiente). Tras su ejecución, al agente se le proporciona una recompensa con la que se calculará la pérdida obtenida.

Esta función de pérdida se basa en la *Ecuación de Bellman* para calcular los valores de Q en cada iteración. Restamos los antiguos valores de Q de la nueva Q calculada y añadimos la recompensa elevando al cuadrado este resultado para calcular la pérdida de un par de estado/acción.

$$L = \frac{1}{2}[r + \gamma(max_{a'}Q(s', a')) - Q(s, a)]^2$$

Esta implementación, además cuenta con un “Replay buffer” para que la red no suministre los estados uno por uno ya que esta implementación haría que los estados estén muy correlacionados por su naturaleza secuencial y que la función se atasque en mínimos locales al observar los estados más recientes. Este buffer almacena las experiencias pasadas <s,a, s’, r> y, al entrenar la red, se tomarán un lote de experiencias pasadas de forma aleatoria. Con esto se suple el problema de la correlación.

Conclusiones

Me ha parecido un proyecto muy interesante ya que solo tenía conocimientos muy básicos de las DQN. Este código es parte de un código de una DQN doble y se ha llevado a cabo la implementación básica de este, eliminando la segunda red.

Los resultados obtenidos con respecto al original son muy parecidos sin tener en el código implementada una segunda red.

Se adjunta un video del funcionamiento del juego implementado. Se puede percibir que necesita más entrenamiento para obtener unos mejores resultados pero como se puede observar algunas veces sobrepasa los 1000 de puntuación y por tanto de recompensa.

Vídeo de prueba

Sobre el código original se ha cambiado que PacMan pueda tener más movimientos aleatorios ya que anteriormente no era capaz de girar nunca a la izquierda y ya sí puede. Además, los caminos que lleva a cabo en las distintas ejecuciones van cambiando para que vaya explorando distintas zonas del mapa.

La peor puntuación conseguida ha sido 210 y la mejor han sido 1280 en todas las pruebas llevadas a cabo.

Referencias

Deep Reinforcement Learning Han - Maxim Lapan.

DQN Tensorflow

Librería baselines de OpenAI.

DQN original.