Indice

Resumen

En este proyecto se desarrolló una implementación del algoritmo NEAT (Neuroevolution Of Augmenting Topologies) en Python 3 y *gym-retro*, con el objetivo de simular y analizar el comportamiento de un agente jugador en el primer nivel del videojuego *Donkey Kong Country* para la consola *Super Nintendo* (SNES).

Se realizaron dos simulaciones del proceso de evolución, y finalmente se obtuvo el comportamiento deseado. Con estos resultados se pudo comprobar el algoritmo NEAT con el aprendizaje no supervisado del videojuego como un entorno de experimentación donde desarrollar algoritmos de inteligencia computacional de nivel humano.

Este proyecto open source se encuentra en https://github.com/BrandonZoft/dkc-ai-bot

Introducción

La evolución es un proceso human en donde ha tratado de imitar a la naturaleza para la solución de problemas. Una forma muy simplista de entender la evolución es la selección natural propuesta por Darwin y una serie de cambios aleatorios que propician la misma ventaja o desventaja de cierto individuo contra sus similares.

Este proceso también fue generalizado e implementado, lo conocemos hoy en día como algoritmos genéticos y plantea evaluar poblaciones de soluciones para seleccionar las más aptas, combinarlas para producir descendientes y mutarlas aleatoriamente para explorar todo el espacio de soluciones posibles.

Implementando el algoritmo genético en nuestro proyecto nos permite seleccionar las poblaciones más aptas que realice el objetivo deseado.

Objetivos

I. Estudiar e implementar el algoritmo NEAT en Python con sus respectivas librerías con el entorno de *Gym-Retro*.
II. Completar el primer nivel del videojuego *Donkey Kong Country* de la consola *Super Nintendo*.
III. Medir y analizar los resultados de la implementación del algoritmo.

Características

NEAT

NEAT es un algoritmo capaz de evolucionar los pesos y las topologías de una población de redes neuronales, utilizando mecanismos de refuerzo para realizar un aprendizaje no supervisado.

NEAT empieza con la mínima topología posible y añade neuronas y enlaces, por medio de dos de sus cuatro operadores de mutación. Cada vez que se añade una nueva estructura al genoma se conoce como *innovación* o *fitness*.

Estas innovaciones se almacenan en una base de datos global. Cuando una nueva neurona o enlace se añade al genoma, se referencia la base de datos y se le asigna un identificador de innovación, que nos dice si ha sido previamente descubierto por otro genoma o si es completamente nuevo. De esta manera, todos los genes de la población están identificados por este identificador de innovación.

Por una parte, cuando un gen con un determinado identificador de innovación se encuentra en ambos padres, el hijo hereda de forma aleatoria uno de ellos. Por otra parte, cuando un gen se encuentra solo en uno de los progenitores, si se trata del que tiene un mayor valor de idoneidad, el hijo lo hereda, si no, es descartado.

gym-retro

gym-Retro es un entorno recreativo para investigaciones de algoritmos de Reinforcement Learning. Fue creado por la compañía de investigación de inteligencia artificial OpenAI sin fines de lucro con el objetivo de promover y desarrollar inteligencia artificial que beneficie a la humanidad.

El entorno de *gym-retro* contiene *data-sets* de mas de 1000 videojuegos para las consolas de Sega Genesis, Game Boy, Super Nintendo, entre otros. Nos permite acceder a las variables de un videojuego, en nuestro caso, la posición de la cámara o la puntuación actual.

En adición a las variables, *gym-retro* permite emular y reiniciar con facilidad un nivel en el caso de que nuestro agente no realice ninguna acción.

Fig. 2. Gym-Retro nos permite acceder a las variables dentro de los videjuegos, como el "score".

Donkey Kong Country

Donkey Kong Country es un juego para la consola Super Nintendo lanzado en 1994. Es un juego multiplataforma en el que la idea principal del mismo es ir pasando una serie de pantallas en diferentes "mundos". Los "mundos" están representados por secciones de lo que sería Donkey Kong Island.

El primer nivel del videojuego es "*Jungle Hijinxs"* como se ve a continuación:

Figura 1. Nivel completo de Jungle Hijinxs

Metodología

Un programa de *machine learning* se diferencia de un programa regular en la forma en que la lógica no está explícitamente definida por el programador. El proyecto será considerado como un programa de aprendizaje automático por refuerzo por los siguientes detalles.

En un sistema de reinforcement learning, esta toma entradas, toma su propia decisión y produce una salida, luego aprende de la recompensa obtenida por la salida y repite todo el proceso una y otra vez. Puede sonar abstracto, pero es más o menos la misma forma en que los humanos aprendemos a hacer las cosas también.

Figura 2. Jungle Hijinxs

En nuestro caso, nuestro personaje o agente (*Donkey Kong*), empieza el proceso sin ni siquiera saber que presionando el botón de la derecha va en dirección a la derecha. Para completar el nivel, una persona ordinaria requiere avanzar a la derecha, saltando obstáculos y recogiendo objetos que aumentan su puntaje. En nuestro caso, la recompensa que se le dará a nuestro agente es cuando va a la derecha de la cámara, y cuando recoge el puntaje.

El entorno de *gym-retro* nos da las variables que cambian constantemente dependiendo de las acciones del usuario. Estos se nos de la siguiente manera:

```
{
  "info": {
    "lives": {
      "address": 8258933,
      "type": "|i1"
    },
    "score": {
      "address": 8258857,
      "type": "|u1"
    }
}
```

Estos son valores que se obtienen del emulador. *"lives"* nos indica la cantidad de vidas de nuestro agente y *"score"* nos indica el puntaje. Solo usaremos el puntaje. Se necesitará por igual la posición x de la cámara para indicarnos cuando avance a la derecha.

Esta variable no viene como predeterminado en *retro-gym*. Me tome la libertad de buscar mapas de direcciones RAM del videojuego y convertirlo en forma decimal, ya que estos existen para modificar las versiones de estos. Se implementa la posición X de esta manera.

```
"x": {
    "address": 8257726,
    "type": "<u2"
  }
```

Así que, cuando nuestro agente avance a la derecha, o agarre objetos que den puntaje (plátanos), se aumentara su *fitness score*.

Esta información, junto con otros conjuntos, es alimentado de forma automática al algoritmo genético de NEAT, toma como datos el **Input** y **Población**. Tenemos una población de 60 por cada generación. Para dar un output en cada proceso. Todo esto proceso ocurre de forma simultánea.

NEAT da como input los diferentes estados de pixeles en la pantalla como se ve en la figura 3, en total son 896 pixeles para representar. La representación se da gracias a OpenCV.

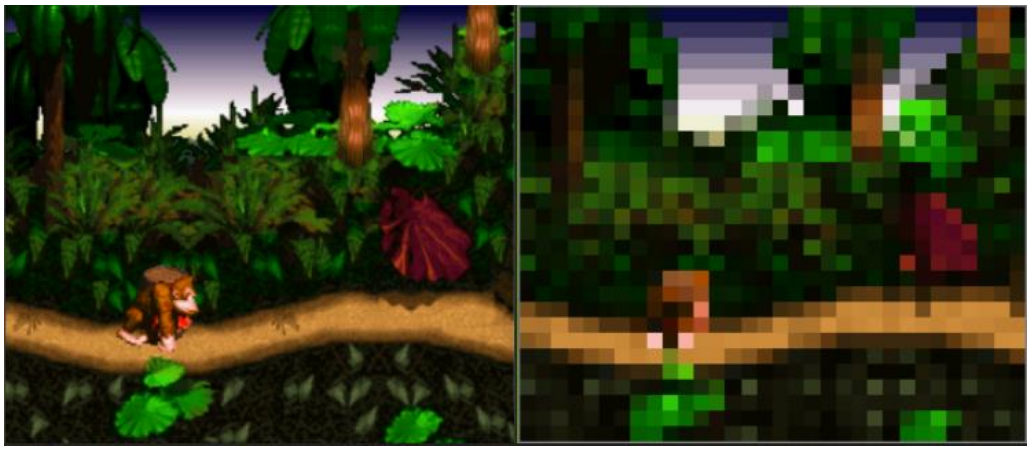

Figura 3. Izquierda: emulador, *derecha:* representación del algoritmo

Como output, el algoritmo tiene como combinación, los 9 botones originales del Super Nintendo. En el experimento, el agente tiene cierto tiempo para aumentar su *fitness score*, o por consecuencia, terminara el proceso y se reiniciara. Todo lo explicado en la metodología se explica por el siguiente proceso de código:

```
while not done:
            xpos = info['x']

            fitness_current += rew * 5

            if xpos > xpos_max:
                fitness_current += 0.25
                xpos_max = xpos

            if xpos_max >= 5120:
                    fitness_current += 1200
```

```
            done = True

        if fitness_current > current_max_fitness:
            current_max_fitness = fitness_current
            counter = 0
        else:
            counter += 1

        if done or counter == 140:
            if fitness_current <= 15:
                fitness_current = 0
            else:
                fitness_current -= 15
            done = True
```

Resultados

Se simulo el aprendizaje dos veces, con una duración aproximada de dos a tres horas cada uno. En la primera simulación, se tomó 9 generaciones para completar el nivel, para un lapso de 2 horas. En la segunda simulación, tomo 12 generaciones en un lapso de 3 horas.

Se encontró que, en las primeras generaciones, el agente no presionaba ningún botón y aumentaba de manera progresiva cuando aumentaba las generaciones.

En las primeras generaciones se detectaron varios problemas, el agente seguía caminando hacia los enemigos, cuando realmente debe de saltar o rodar para evadirlos. A la mitad de estas generaciones, la mayoría se quedaban en este punto mostrado en la figura 4.

Figura 4

Este punto fue el más difícil de superar para el agente, en muchos casos se presentaba que el agente no saltaba y se reiniciaba el proceso. Incluso si lo superaba, tendría que saltar de manera precisa a los tres enemigos de la figura.

Una acción que implemento el agente en las últimas generaciones es rodar y saltar cuando ve un enemigo. Cuando el personaje *Donkey Kong* rueda, se la una velocidad rápida por corto tiempo. Esto es un mecanismo que usan los llamados *speedrunners* (llamados así para las personas que completan niveles de videojuegos de la manera más rápida posible) para completar el nivel más rápido, pero el agente lo usa de manera eficiente para lidiar con enemigos.

En las últimas generaciones, el obstáculo que presentaba era un enemigo que se tenía que evadir e inmediatamente saltar como se representa en la figura 5. Esto se presentó desde las generaciones 6, esto significa que le tomo 3 a 6 generaciones para superar este simple obstáculo, puesto que cuando superas este obstáculo, llegas al final del nivel.

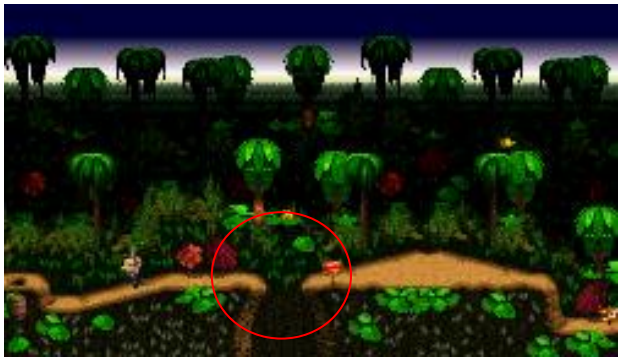

Figura 5

Conclusión

La conclusión clave del proyecto es comprender cómo un algoritmo de aprendizaje automático es más similar a cómo los humanos aprenden a completar tareas que un programa de procedimiento.

En un programa de procedimiento, la toma de decisiones está codificada en el programa a través de sentencias dadas por un programador. Esto sería, por ejemplo, el programador que identifica las amenazas en el juego de Super Mario e ingresa declaraciones como: si se detectan amenazas, realice esta acción en particular o continúe.

La principal diferencia entre este enfoque de procedimiento y un enfoque de aprendizaje automático es que, en última instancia, el programa informático tiene autonomía para tomar sus propias decisiones. Esto es importante de entender porque el programa de aprendizaje automático puede usarse en otras tareas similares sin tener que ser reprogramado, aunque no está limitado al juego de *DKC* puesto que los niveles consecuentes son más complejos (nadar abajo del agua con movimiento más lento).

Un juego en el que el entrenamiento se da en el primer nivel y usar el entrenamiento para usarlos en otros niveles podría ser en el juego original de Mario Bros, puesto que el nivel dos es similares al nivel uno.

Este comportamiento es muy similar a cómo un ser humano se volvería mejor y mejor jugando juegos diferentes, aunque no reiniciamos cada vez que cometemos un error como en el experimento, nosotros aprendemos de manera mucho más rápida porque usamos nuestras experiencias anteriores para aprender más rápido.

Referencias

- Stanley, K. O., & Miikkulainen, R. (2002). Efficient evolution of neural network topologies. In Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600) (Vol. 2, pp. 1757-1762). IEEE.
- Sempere Tortosa, M. L., Gallego-Durán, F. J., Llorens Largo, F., Pujol, M., & Rizo, R. (2005). Un entorno de aprendizaje neuro evolutivo: *Screaming Racers*.
- Luis E. Ramirez. (2014). Using Artificial Neural Networks to Play Pong. Swarthmore College Computer Science Department
- Rodrigo Gamba Lavín (2019). Implementación del algoritmo NEAT (neuroevolución por topologías aumentadas) para el control de un sistema caótico. Universidad Nacional Autónoma De México.
- Understanding basics of machine learning through Super Mario. (2017). Towards Data Science: https://towardsdatascience.com/understanding-basics-of-machine-learning-through-super-mario-5ed93c7d587d
- Lucas Thompson. (2019) Artificial intelligence in Python Retrieved from https://www.youtube.com/channel/UCLA_tAh0hX9bjl6DfCe9OLw
- CodeReclaimers, Neat-Python. (2018). Repositorio GitHub: https://github.com/CodeReclaimers/neat-python
- OpenAI, retro. (2019). Repositorio GitHub: https://github.com/openai/retro