Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Neuroevolución

Jugando Snake

Alumno: Daniel Soto
Profesor: Alexander Bergel
Auxiliar: Juan-Pablo Silva
Ayudantes: Alonso Reyes
Gabriel Chandia

Fecha de entrega: 3 de enero de 2019
Santiago, Chile

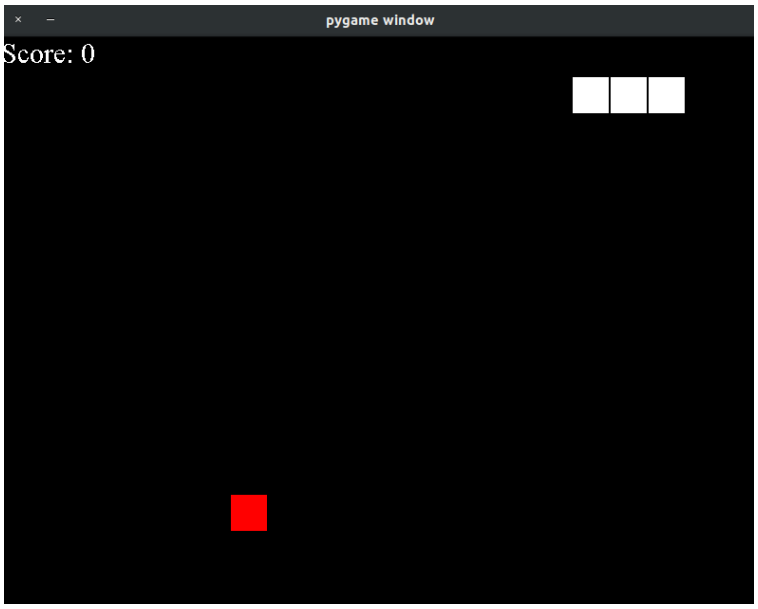


Figura 1: El juego Snake en acción.

Repositorio

El repositorio donde se llevó a cabo el desarrollo de esta tarea se encuentra aquí.

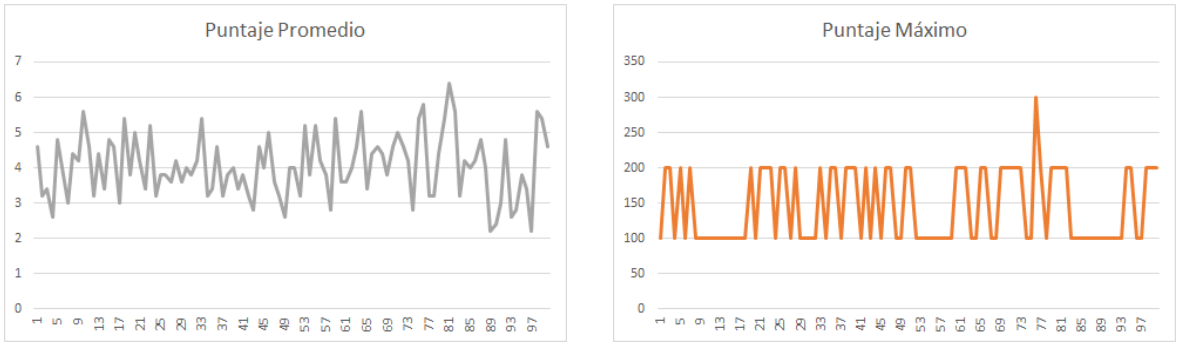
1. Problema

El problema elegido consistía en crear una red neuronal que fuese capaz de jugar el clásico juego Snake (Figura 1). La red sería generada utilizando Neuroevolución, que toma una red neuronal como un individuo que se reproduce, con la función de fitness definida exclusivamente como el puntaje que logra obtener la red en el juego.

Las redes que intentan resolver el problema, fueron diseñadas como redes con una capa oculta de 8 neuronas. Éstas reciben como inputs el contenido de cada celda en el juego, siendo -1 cuando está la serpiente allí, 0 cuando está vacía y 1 cuando hay una fruta.

2. Descripción del Programa

Se puede comenzar el proceso de aprendizaje ejecutando el archivo `ai_play.py`, usando Python con una versión `>=3.6`. Esto desplegará una ventana, donde se mostrará una visualización de lo que hace cada individuo de la población, mientras se calcula su fitness (es decir, el individuo juega). Se puede cambiar la forma de la red modificando el archivo `network_individual.py`, cambiando como se instancia la red, y como se reproduce un individuo. Se puede cambiar el número de individuos cambiando el valor de la variable `N` en `ai_play.py`. En este último archivo, también se puede cambiar la función de fitness utilizada por el algoritmo, y la función de filtrado de la población. Como en la tarea anterior, se entregan funciones de filtrado en el archivo `filtering.py`,



(a) (b)

Figura 2: Resultados experimentales del algoritmo.

documentadas para saber cómo usarlas.

El algoritmo es una extensión de los dos algoritmos implementados en las tareas 1 y 2. Se usó la red neuronal como uno de los parámetros almacenados por un individuo. Además se agregaron métodos para transformar una red a una lista de neuronas discretas, para poder implementar las operaciones genéticas de los individuos.

El juego fue creado con `pygame` y se puede jugar ejecutando el archivo `play.py`.

3. Evaluación

Se ejecutó el programa por 100 generaciones, grabando el fitness del mejor individuo, y el fitness promedio de la población. Los resultados obtenidos se pueden ver en la figura 2.

4. Discusión

Es claro que el algoritmo no logró los resultados que se buscaban, pues no hay una mejora visible, y la población sigue igual de inestable en las últimas generaciones, que como comenzó.

Una posible explicación de este mal rendimiento es que el input no era el apropiado para una red como la usada. Es decir, usar una red tan pequeña para un input tan grande puede ser muy perjudicial para el proceso de aprendizaje.

Posibles maneras de arreglarlo hubiesen sido utilizar una red más compleja, con más capas intermedias, o cambiar el input para que sean menos variables.

Para mantener la eficiencia de la red, y que esta puede ejecutarse entre las frames de un juego, probablemente el mejor acercamiento sería resumir la información que se le entrega, para reducir el tamaño del input. Esto se podría hacer entregándole a la red a que distancia de la serpiente hay un peligro o una fruta, en las cuatro direcciones en las que se puede mover, junto a la dirección en la que se está moviendo la serpiente (representada por dos números). De este modo se tendrían tan solo 6 variables, una para cada dirección en la que se puede mover la serpiente, y dos para la dirección en la que se está moviendo la serpiente. Además, se pueden agregar más capas intermedias, que gracias al input más pequeño no debiesen afectar mucho la velocidad de procesamiento.