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

Tarea 3

Proyecto: Simulador de snake con neuroevolución

Alumna: Romina Romero Oropesa

Profesor: Alexandre Bergel

Auxiliares: Juan Pablo Silva

Ayudantes: Alonso Reyes Feris
Gabriel Chandía

Fecha de entrega: 30 de diciembre de 2018
Santiago, Chile

Índice de Contenidos

Lista de Figuras

Lista de Códigos

Problema: Simulacro de snake

El juego

El juego snake consiste en controlar una larga y delgada criatura similar a una serpiente, que viaja recogiendo alimentos. Cada vez que come un alimento, la criatura crece y se suma puntos. El juego se pierde cuando la serpiente choca con su propia cola, o con alguna pared.

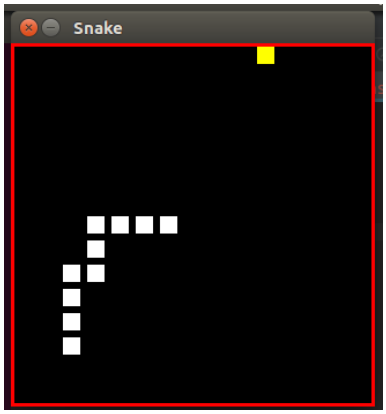

Figura 1: Juego snake

Para este proyecto, el juego tiene las siguientes características:

- El tablero es de 15x15 posiciones.
- Se inicia con una serpiente tamaño 2.
- Si choca con el borde, pierde. Si choca con su cola, pierde.
- Solo aparece un alimento a la vez. Cuando es comido, aparece uno nuevo.

Para ejemplificar, se incluye una implementación simple del juego en el archivo *play_game.py*. Ejecutar el siguiente comando:

Código 1: Ejecutar juego de ejemplo

```
1 python3 play_game.py
```

Las paredes se representan con color rojo, la serpiente con color blanco, el alimento amarillo y el fondo negro.

Modelado

La red neuronal que modela el problema, contiene 6 entradas, una capa oculta con 10 neuronas y 3 salidas.

Las entradas son las siguientes:

- En la dirección actual, distancia a la comida (si no hay comida en mi dirección, se setea el tamaño del tablero).
- En la dirección actual, distancia al obstáculo más próximo. Se considera obstáculo algún segmento de snake y los bordes.
- Si roto en sentido antihorario, distancia a la comida.
- Si roto en sentido antihorario, distancia al obstáculo más próximo.
- Si roto en sentido horario, distancia a la comida.
- Si roto en sentido horario, distancia al obstáculo más próximo.

Las 3 salidas representan:

- Mantener dirección.
- Rotar en sentido antihorario.
- Rotar en sentido horario.

El mayor es quien indica que movimiento debo hacer a continuación.
La función de fitness inicia un nuevo juego. El resultado es el siguiente:

$$fitness_r = tamano\_snake + \frac{desplazamiento\_antes\_de\_perder}{225} \tag{1}$$

Función de fitness.

La función finaliza el juego si snake ha hecho 225 desplazamientos sin crecer (se considera que ha entrado en un loop). Se premia el tamaño de la serpiente, y su capacidad de mantenerse viva.

Descripción del programa

Instalación

El código de la implementación se encuentra en el repositorio de github https://github.com/romina-romero/redes_neuronales_2018_2. El lenguaje utilizado es **python 3**. La tarea se encuentra en la carpeta tarea_3.

Para poder ejecutar los tests y hacer uso de esta implementación, se debe instalar el paquete numpy, matplotlib de python3 con una terminal:

Código 2: Instalación de dependencias.

```
pip install numpy matplotlib scipy pygame
```

Además, será neesario tener instalado el paquete python3-tk. Por ejemplo, para Ubuntu linux instalar asi:

Código 3: Instalación de python3-tk.

```
sudo apt-get install python3-tk
```

En https://pip.pypa.io/en/stable/installing/ se explica como instalar pip.

Implementación de neuroevolución

Se utiliza la clase NeuralNetwork implementada en la tarea 2, agregando un método para transformar a la red en un vector de neuronas, y otro para decodificar y obtener la red nuevamente. Para correr los unit test de la clase, ejecutar:

Código 4: Unit-testing

```
python3 -m unittest neural_network/NeuralNetwork.py
```

Para neuroevolución, se implementan las clases Individual y Population.

Cada instancia de la clase Individual contiene una red neuronal. Esta clase implementa el método crossover, en que las redes neuronales son transformadas a vectores de neuronas, luego combinadas. La cantidad de neuronas de cada parte es proporcional al fitness. La mutación consiste en un cambio aleatorio de algún peso o bias.

La clase Population implementa la selección y reproducción. Para la selección, se ordena de mayor a menor fitness y se seleccionan los k mejores, donde k es la cantidad de seleccionados. La reproducción toma pares aleatorios y los combina para crear la nueva población. Es posible además indicar una cantidad de población elite. Este número indica cuántos individuos se conservan de la población original (ordenados por fitness).

Implementación simulador de snake

La clase Snake representa a la serpiente, la clase Segment a un segmento de la serpiente, la clase Food a la comida y la clase Board al tablero. Estas clases se usan para hacer el entrenamiento. La clase Board implementa el método *fordward*, que permite hacer avanzar a snake en el tablero, además del método *rotate* que cambia la dirección de snake. Implementa además los métodos *food_distance* (distancia a la comida) y *obs_distance* (distancia al primer obstáculo). Snake mantiene el parámetro *alive* que indica si la serpiente sigue o no viva.

Se provee además de un simulador, que permite usar una red neuronal para jugar. La función simulare se encuentra en el archivo helper.py.

El archivo train_snake.py entrena a la red neuronal mediante neuroevolución con snake. Al finalizar, se visualizará una simulación con la mejor red neuronal encontrada, y luego se despliega el gráfico de máximos, promedios y desviaciones estándar por generación.

Para ejecutar el entrenamiento y simulación:

Código 5: Entrenamiento y simulación de snake

```
python3 train_snake.py
```

Para modificar los hiperparámetros, en el sector superior del archivo train_snake.py se encuentran las variables globales. Cambiarlas según se requiera:

Código 6: Hiperparámetros.

```
POPULATION_SIZE = 100
GENERATIONS = 1000
SELECTION_SIZE = 20
MUTATION_RATE = 0.3
ELITE = 5
```

- **POPULATION_SIZE**: cantidad de individuos por generación.
- **GENERATIONS**: número de generaciones de entrenamiento.
- **SELECTION_SIZE**: cantidad de seleccionados en proceso de selección.
- **MUTATION_RATE**: porcentaje de individuos que mutarán.
- **ELITE**: número de individuos que se conservarán de una generación a otra según su fitness.

Evaluación

Para la evaluación, se usaron los valores por defecto indicados en la sección anterior. Es decir, se entrenó con 1000 generaciones de 100 individuos. Los resultados obtenidos son los siguientes:

- Máximo valor de fitness: 7.058
- Tiempo de ejecución: 59 minutos

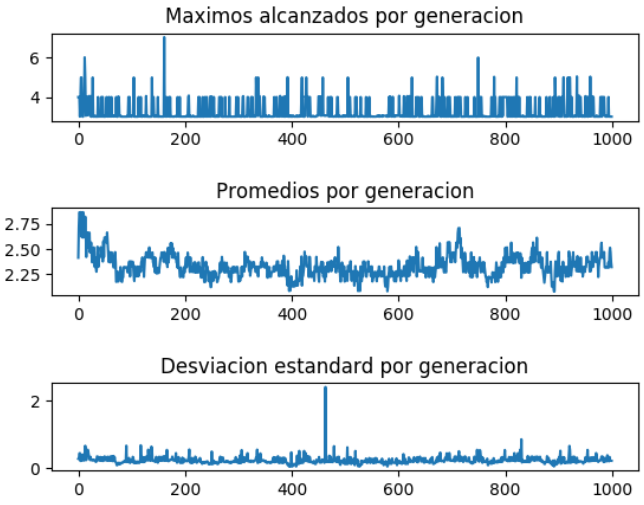

Figura 2: Estadísticas por generación

Discusión

Se considera que en esta implementación, neuroevolución no resuelve el problema a cabalidad, pues se obtuvo un largo de serpiente de 7 en un tablero de 15 x 15, la cual es aun pequeña.

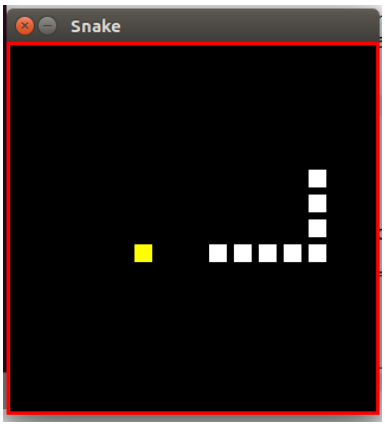

Figura 3: Snake de largo 7

Además, en el gráfico se observan algunos peaks de fitness, pero no hay un incremento en el tiempo.

Se cree que para que funcione mejor, sería necesario pulir la función de fitness.

En este ejemplo, la función de fitness ejecuta un solo juego. Las comidas aparecen de forma aleatoria, por lo que el ejecutar solo una vez el juego para evaluar, agrega un sesgo al resultado. Como mejora, es buena idea ejecutar una cantidad de veces el juego, y promediar los puntajes.

Otro problema que se encontró, es que el puntaje entregado por mantenerse viva, tienden a hacer a la serpiente entrar en loops. Para corregirlo, en este ejemplo se detiene de manera abrupta al hacer 255 movimientos sin alcanzar una comida. Pero sería bueno desincentivar los loops castigando, por ejemplo, el pasar por el mismo lugar sin alcanzar una nueva comida.