LA UNIVERSIDAD EN INTERNET

COMPUTACIÓN BIOINSPIRADA

ACTIVIDAD 3

Algoritmos Evolutivos

Utilizar algoritmos evolutivos para comprender la mecánica que hay detrás

Profesor: Javier Martínez Torres

Ernesto González Pradas
ernesto.gonzalez023@comunidadunir.net

Índice

Contenido

Índice....1
Introducción....2
Buscar un problema para resolver con algoritmo evolutivo....3
Buscar una librería que implemente algoritmo evolutivo....4
Realizar diversas ejecuciones y determinar el impacto....6
Conclusión....7
Bibliografía....8

Introducción

El objetivo de este laboratorio es comprender como funcionan los algoritmos evolutivos. Para llevar a cabo esta tarea, vamos a buscar un problema que necesite ser optimizado mediante uno de estos algoritmos, posteriormente buscaremos una implementación de cualquiera de los algoritmos expuestos y finalmente analizaremos el impacto que tiene modificar los parámetros de entrada sobre la solución final.

El problema será un ajuste de pesos en una función matemática dada y la implementación del algoritmo evolutivo la buscaremos en lenguaje Python.

Buscar un problema para resolver con algoritmo evolutivo

A continuación, mostraremos el problema que hemos seleccionado para resolver con el algoritmo evolutivo.

Este problema busca encontrar de forma óptima, un conjunto de pesos que satisfaga la siguiente función:

$$y = w1x1 + w2x2 + w3x3 + w4x4 + w5x5 + w6x6$$

Siendo (x1, x2, x3, x4, x5, x6) = [4, -2, 3.5, 5, -11, -4.7] y y=44, es decir, vamos a buscar con nuestro algoritmo evolutivo cuanto tiene que valer (w1, w2, w3, w4, w5 y w6) para que la función sea y = 44.

$$44 = w1 * 4 + w2 * (-2) + w3 * (3.5) + w4 * 5 + w5 * (-11) + w6 * (-4.7)$$

Como podemos observar es un problema sencillo de entender, pero muy intuitivo a la hora de ver cuales son los pesos seleccionados en cada una de las iteraciones que va a realizar nuestro algoritmo evolutivo en función del número de generaciones, genes, el tipo de selección si le ponemos factor de mutación o no, etc.

Buscar una librería que implemente algoritmo evolutivo

Para el desarrollo de esta actividad, después de mirar varias librerías en diversos lenguajes, nos hemos decantado por utilizar la librería PyGAD escrita en Python.

PyGAD es una libreria de código abierto que suministra un algoritmo genético y además permite optimizar los algoritmos de aprendizaje automático. Funciona con Keras (que es otra librería de código abierto escrita en Python con el propósito de permitir que las redes neuronales se desarrollen rápidamente) y con PyTorch (librería open source enfocada en la realización de cálculos numéricos mediante programación de tensores).

A continuación, vamos a ver algunos de los parámetros de entrada que nos permite modificar esta librería y que toquetearemos en el siguiente apartado para ver el impacto que tiene en los resultados:

- **Num_generations:** es el número de generaciones.
- **Num_parents_mating:** es el número de soluciones que se seleccionarán como padres en el grupo de apareamiento.
- **Fitness_func:** acepta una función que recibe dos parámetros (una única solución y su índice en la población) y devuelve el valor fitness de la solución.
- **Initial_population:** se introduce una población inicial definida por el usuario. Si el usuario no introduce ninguna, este parámetro por defecto será “None”. Si es “None”, PyGAD por defecto generará una población inicial en base a sol_per_pop (número de soluciones en la población) y num_genes (número de parámetros de la función).
- **Init_range_low:** será el valor inferior del rango aleatorio del que se seleccionan los valores de los genes en la población inicial. Por defecto es -4.
- **Init_range_high:** igual que init_range_low pero siendo el valor superior.
- **Parent_selection_type:** tipo de selección de padres.
- **Keep_parents:** Si es 0, significa que ningún padre de la población actual se utilizará en la siguiente población y si es -1, significa que todos los padres de la población actual si se utilizarán en la siguiente población. Si se selecciona un valor > 0, dicho valor especificado se refiere al número de padres de la población actual que se utilizarán en la siguiente población.
- **Crossover_type:** tipo de operador de cruce. Si es “None” se omite el paso de cruce y por tanto no se creará ninguna descendencia en las siguientes

generaciones. La siguiente generación utilizará las soluciones de la población actual.

- **Mutation_type:** tipo de operador de mutación. Si este parámetro se encuentra como “None” e omite el paso de mutación por lo que no se aplican cambios a la descendencia creada mediante la operación cruce. La descendencia se utilizará sin cambios en la siguiente generación.
- **Mutation_percent_genes:** porcentaje de genes a mutar que por defecto es la cadena ‘default’ que significa 10%. Este parámetro no tiene acción si existe alguno de los 2 parámetros mutation_probability o mutation_num_genes.

Realizar diversas ejecuciones y determinar el impacto

Ahora vamos a ejecutar nuestro algoritmo varias veces, modificando los atributos de entrada y ver que sucede. Como hemos descrito en el apartado anterior vamos a modificar las entradas más interesantes del algoritmo.

A continuación, se adjunta el fichero Excel con todas las pruebas realizadas y con que parámetros se han realizado las pruebas. Estas pruebas han constado de una serie de batería de ejecuciones con cinco cambios en los valores de entrada. Después pasaremos a comentar los resultados:

pruebasAlgoritmo.xlsx

Como podemos ver en la primera prueba, al tener unos valores de "init_range_low" y "init_range_high" entre (-2, 5) respectivamente, los mejores parámetros de la solución se encuentran entre esos rangos. En la cuarta y quinta batería de ejecuciones, vemos que, al variar estos parámetros, dichas soluciones nos salen en el rango introducido (-20, 50).

Analizando todas las baterías de ejecuciones, uno de los parámetros más importantes para encontrar los parámetros de la mejor solución y un valor de la función de evaluación (que nos permite saber la calidad de los individuos de la población) es, que el algoritmo **mute** (en nuestro caso le hemos dicho que mute de forma aleatoria). Se puede apreciar que la salida prevista basada en la mejor solución es prácticamente la solución buscada (y = 44) con un error de 10^{-3}.

Si le decimos al algoritmo que nos calcule las mejores soluciones de los pesos añadiendo 500 números de generaciones en vez de 50, los resultados se alejan bastante de la función optimizada que estamos buscando (algunos ejemplos que se pueden ver en el Excel y ≈ 70, y ≈ 36). Sin embargo, en el momento que le volvemos a introducir como parámetro de entrada que el tipo de mutación sea aleatoria en vez de ninguna, volvemos a obtener resultados muy optimizados con un error de 10^{-4}.

Estos resultados, están estrechamente relacionados con lo visto en el tema sobre algoritmos evolutivos visto en clase, ya que una de las principales características de las estrategias evolutivas es el operador de mutación. Este operador facilita tanto la exploración como la explotación del espacio.

Conclusión

Como hemos podido ver con las distintas pruebas uno de los factores más importantes es el operador de mutación. Y es que, aunque tengamos una población inicial muy grande (que nos permitiría explorar más el espacio) lo que realmente nos ayuda a encontrar una optimización de nuestra solución es la capacidad de mutación de los mejores resultados, ya que no solo exploramos más si no que también explotamos más.

Esta es una de las principales diferencias con respecto a los algoritmos genéticos vistos en el tema tres. Y es que, aunque en internet cuando buscamos información sobre librerías de algoritmos evolutivos se utilizan ambos términos (evolutivo y genético) indistintamente, pero esta es una de las principales diferencias. Mientras que en los algoritmos genéticos el operador de mutación era algo opcional y que incluso muchas veces nos limitaba la explotación del algoritmo ya que solo favorecía la exploración, en los algoritmos evolutivos se aplica siempre sobre soluciones escogidas.

Bibliografía

Gad, A. F. (2020). *pygad*. Obtenido de https://pygad.readthedocs.io/en/latest/

Keras. (1 de 1 de 2022). *Keras*. Obtenido de https://keras.io/

pytorch. (1 de 1 de 2022). *pytorch*. Obtenido de https://pytorch.org/