Algoritmo AHA

1. ¿Qué es un algoritmo de optimización bioinspirado?

   - Un algoritmo de optimización bioinspirado es un método computacional que toma inspiración de procesos naturales, como la evolución, el comportamiento de animales o la dinámica de ecosistemas, para resolver problemas de optimización. Este algoritmo imita estrategias eficientes encontradas en la naturaleza para encontrar soluciones optimas en diferentes contextos, como en la inteligencia artificial y el aprendizaje automático.

2. Fundamentos del AHA:

   2.1. **Habilidades de vuelo del colibrí**: En el algoritmo, el "vuelo" representa los movimientos dentro del espacio de búsqueda. Los colibríes pueden moverse en direcciones distintas (axial, diagonal, omnidireccional), lo que en términos de optimización significa que el algoritmo puede explorar soluciones desde diferentes ángulos y con distintas estrategias. Esto es para ayudar a mejorar la capacidad de búsqueda y evitar quedar atrapados en óptimos locales.

   2.2. Estrategia de forrajeo: Este representa la forma en que se busca soluciones óptimas, estas pueden ser:

   - Guiado: El algoritmo sigue una dirección basada en la mejor solución encontrada hasta el momento
   - Territorial: Busca soluciones en una región especifica antes de expandirse a otras áreas.
   - Migración: Cambia completamente la estrategia cuando no encuentra mejoras en la región actual.

   2.3. Memoria y toma de decisiones en el AHA: El algoritmo almacena información sobre soluciones exploradas previamente, lo que le permite aprender y evitar evaluar constantemente las mismas soluciones. La toma de decisiones se basa en la comparación de resultados previos y en la adaptación de nuevas estrategias de búsqueda

3. Comparación con PSO (Optimización por enjambre de partículas) y ABC (Algoritmo de la colonia de Abejas):

   3.1. PSO
   - Esta basado en el comportamiento de enjambres de partículas.

- Cada partícula representa una solución y se mueve en función de su experiencia y la del grupo.
- Se centra en mejorar la solución global y la mejor individual.

3.2. ABC

- Modela la forma en que las abejas buscan alimento
- Se divide en abejas exploradoras, obreras y observadoras que evalúan y comunican la calidad de diferentes fuentes de alimento (estas son las posibles soluciones)
- Balancea la exploración de nuevas soluciones con la explotación de buenas soluciones encontradas.

Ejercicio reflexivo

- ¿Cómo crees que el equilibrio entre exploración y explotación en AHA puede mejorar el rendimiento de un modelo de IA?

  - **Exploración**: Se refiere a la capacidad del algoritmo de buscar soluciones nuevas en diferentes partes del espacio de búsqueda.
  - **Explotación**: Es la capacidad de mejorar soluciones dentro de una región prometedora.

Un buen equilibrio significa que el algoritmo no se queda atrapado en soluciones subóptimas (porque explora lo suficiente) ni pierde tiempo probando demasiadas opciones innecesarias (porque también explota las mejores soluciones encontradas).

En modelos de inteligencia artificial, este equilibrio puede mejorar la capacidad de ajuste de los modelos, evitando caer en mínimos locales y encontrando configuraciones óptimas más rápido.

Implementación de AHA en Python

1. Explicación Matemática

❖ **Inicialización del Algoritmo**

Sea un espacio de búsqueda definido por una función objetivo $f(x)$ donde x representa las soluciones candidatas.

- Se genera una población inicial N colibríes con posiciones aleatorias en el espacio de búsqueda:

$$X_i = (x_i1, x_i2, \dots., x_{id}) \text{ para } i = 1,2, \dots, N$$

Donde d es la dimensión del problema.

- Se evalúa la calidad de la solución usando la función objetivo $f(X_i)$.

❖ Movimiento de los colibríes

Los colibríes pueden moverse utilizando tres tipos de vuelos:

- Movimiento Axial (Exploración local - Refinamiento de solución):

$$X_i^{t+1} = X_i^t + r * (X_{besi}^t - X_i^t)$$

Donde r es un numero aleatorio en [0,1] y X_{best}^t es la mejor solución global.

- Movimiento Diagonal (Exploración global - Evitar óptimos locales):

$$X_i^{t+1} = X_i^t + \beta * (X_j^t - X_k^t)$$

Donde X_j^t y X_k^t don dos colibríes seleccionados aleatoriamente y β es un factor de movimiento.

- Movimiento Omnidireccional (Exploración aleatoria - Diversificación):

$$X_i^{t+1} = X_i^t + \alpha * \mathrm{N}(0,1)$$

Donde $N(0,1)$ es una distribución normal con media 0 y varianza 1, y α es un parámetro de ajuste.

1.2. Pseudocódigo

2. **Inicialización**:
   - Definir el espacio de búsqueda y los parámetros del algoritmo.
   - Generar una población inicial de soluciones (colibríes).
3. **Iteraciones hasta convergencia**:
   - Evaluar la aptitud de cada colibrí.
   - Aplicar los movimientos de vuelo: axial, diagonal u omnidireccional.
   - Usar estrategias de forrajeo para actualizar posiciones.
   - Usar memoria para recordar buenas soluciones.
4. **Criterios de parada**:
   - Si el número de iteraciones ha sido alcanzado o la mejora en la solución es mínima, terminar.
   - Retornar la mejor solución encontrada.

2. Parámetros del algoritmo

- Criterio de Parada
  - Define cuando el algoritmo debe detenerse. Puede ser un número máximo de iteraciones, un umbral de mejora mínima o una condición de convergencia.
  - Impacto: Un criterio de parada estricto puede evitar encontrar la mejor solución, mientras que uno muy largo aumenta el tiempo de ejecución.
- Tamaño de población de Soluciones
  - Indica cuantas soluciones se maneja simultáneamente.
  - Impacto: Un tamaño mayor explora más opciones, pero aumenta el costo computacional. Un tamaño menor acelera la ejecución, pero puede llevar a soluciones subóptimas.
- Factor de exploración y explotación
  - Controla el balance entre explorar nuevas soluciones y mejorar las existentes
  - Impacto: Una mala elección puede hacer que el algoritmo optimice valores irrelevantes o se estanque en soluciones no ideales.
- Mecanismo de Actualización de la mejor solución
  - Define como se comparan y almacenan las mejores soluciones encontradas.
  - Impacto: Un buen mecanismo evita perdidas de soluciones optimas y ayuda a una convergencia más rápida.

2.1. Impacto en la optimización

- El ajuste de estos parámetros afecta la capacidad del algoritmo para encontrar soluciones óptimas en un tiempo razonable. Si los valores están bien calibrados, el algoritmo puede **encontrar soluciones de alta calidad** de manera eficiente.

Pregunta de Inferencia

¿Cómo podrías modificar el AHA para que sea más eficiente en problemas de alta dimensionalidad?

- Para hacer que el algoritmo AHA sea más eficiente en problemas muy complejos, se pueden reducir los datos eliminando información innecesaria (PCA o selección de características), usar búsqueda inteligente (búsqueda local adaptativa o enfriamiento simulado) para enfocarse en las mejores opciones, procesar varias soluciones al mismo tiempo con múltiples procesadores, ajustar sus reglas automáticamente según la etapa del proceso y combinarlo con otros métodos como los algoritmos genéticos o PSO. Con estos cambios, el algoritmo encontrará mejores soluciones en menos tiempo sin perder precisión.

Aplicaciones del AHA en inteligencia Artificial

Optimización de Hiperparametros en Redes Neuronales

1. Uso del AHA para encontrar los mejores valores de hiperparámetros en modelos de aprendizaje profundo.

   - El algoritmo AHA se puede utilizar para ajustar automáticamente los hiperparametros de una red neuronal, como la tasa de aprendizaje, el numero de neuronas de cada capa o tipo de función de activación. AHA evalúa diferentes configuraciones y selecciona las mejores en lugar de probar combinaciones al azar, esto mejora el rendimiento del modelo sin intervención manual.

2. Comparación con métodos como Grid Search y Bayesian Optimization

- Grid Search: Este prueba todas las combinaciones posibles de hiperparámetros en una cuadricula predefinida, lo que garantiza encontrar una buena solución pero es muy costoso en tiempo y recursos.

- Bayesian Optimization: Utiliza modelos probabilísticos para predecir que combinaciones de hiperparametros darán mejores resultados, optimizando el proceso con menos evaluaciones.

Conclusión

- AHA puede ser más eficiente que Grid Search porque no prueba todas las combinaciones, y puede ser más flexible que Bayesian Optimization al adaptarse dinámicamente durante la búsqueda.

Pregunta Reflexiva

¿Por qué la capacidad de memoria del AHA puede ser útil en la optimización de hiperparámetros?

- La capacidad de memoria del AHA es útil en la optimización de hiperparámetros porque le permite **recordar configuraciones anteriores y sus resultados**, evitando probar combinaciones repetidas o ineficientes. Esto mejora la eficiencia del proceso al centrarse en opciones prometedoras y acelerar la convergencia hacia los mejores valores. Además, al aprovechar la información previa, el AHA puede ajustar su estrategia de búsqueda de manera más inteligente, reduciendo el tiempo y los recursos necesarios en comparación con enfoques como **Grid Search**, que evalúan todas las combinaciones sin aprendizaje previo.

Selección de características en Machine Learning

1. Cómo el AHA puede seleccionar un subconjunto óptimo de características para mejorar la precisión de modelos.

    - El AHA puede seleccionar un subconjunto óptimo de características mediante un proceso de búsqueda inteligente que evalúa diferentes combinaciones de variables y el impacto que tienen en la presión del modelo . Este utiliza su capacidad de memoria para recordad que conjuntos de características han funcionado mejor y ajustar su estrategia en cada iteración.

-