***Escriba el pseudocódigo de algoritmos genéticos. Explique brevemente las principales diferencia entre AG binarios y reales. De 3 casos de aplicación de AG y sus respectivas fuentes (paper, autor, fecha), con breve explicación. Como lo implementaría usando programas disponibles open-source (cita y breveexplicación).***

**Algoritmos genéticos**

Los algoritmos genéticos (AG) son técnicas de búsqueda y optimización que se inspiran tanto en la teoría de la evolución de Darwin como en los principios de la genética natural. De este modo, la lógica que utilizan para resolver problemas complejos de optimización es similar a la selección natural.

La idea es simple, partiendo de una población de posibles soluciones (o individuos) a un problema que compiten entre sí, aquellos que son mejores son los que se reproducen combinando información. De esta manera, se logran introducir mutaciones paulatinas que mantienen la diversidad de la población pero que, con el tiempo, logra evolucionar hacia soluciones mejores.

**Conceptos clave:**

* **Individuos o cromosomas:** representan a cada posible solución al problema, pueden estar codificados de manera binaria (0 y 1) o de manera real (con números continuos)
* **Gen:** es cada uno de los componentes del cromosoma, por ejemplo, un número
* **Población:** es el conjunto de individuos en una generación, el tamaño de esta afecta a la diversidad y velocidad de convergencia
* **Fitness (o aptitud):** es la función que mide qué tan buena es una solución, por lo que es el “criterio de supervivencia” bajo el cual se define quienes tienen más probabilidad de reproducirse
* **Operadores genéticos:** 
  + **Selección:** elige quiénes se van a reproducir
  + **Crossover:** combina pares de individuos para generar la descendencia
  + **Mutación:** introduce los cambios aleatorios en los genes que mantienen la diversidad
* **Elitismo**: aquellos individuos de una generación que son mejores son copiados a la siguiente garantizando que la calidad no empeore

**Pseudocódigo de un algoritmo genético:**

La dinámica de un algoritmo genético es la siguiente:

1. Se genera aleatoriamente a una población de posibles soluciones
2. Se evalúa la aptitud o fitness de cada uno de los individuos que la componen
3. Se seleccionan a los más aptos para reproducirse
4. Se da la reproducción donde se aplican crossover y mutación para poder crear individuos nuevos
5. Se forma la nueva generación incluyendo élites si aplica

Este ciclo se itera las veces que sean necesarias hasta lograr un criterio de parada que puede ser una cantidad puntual de generaciones o un fitness mínimo, por ejemplo.

De este modo, podemos definir el siguiente pseudocódigo:

**Entrada:**

* Función de fitness f(x) a maximizar (o minimizar)
* Espacio de búsqueda y codificación de individuos
* Tamaño de población N, prob. de cruce pc, prob. de mutación pm
* Criterio de parada (nº de generaciones G, umbral de fitness, etc.)

**Algoritmo:**

1. Inicializar población P0 con N individuos al azar
2. Evaluar fitness de cada individuo en Pt
3. (Mientras no se cumpla el criterio de parada)
   1. Selección: elegir padres de Pt según fitness
   2. Reproducción:
      1. Cruce con prob. pc → generar descendencia
      2. Mutación con prob. pm → perturbar genes
   3. Formar población candidata C a partir de descendencia (opcional: elitismo, reemplazo generacional/estacionario)
   4. Evaluar fitness de C
   5. Actualizar Pt+1 ← C (y élites si aplica)
   6. t ← t + 1
4. Devolver el mejor individuo encontrado

**Salida:**

* Mejor solución y su fitness

**Diferencias principales entre algoritmos genéticos reales y binarios**

Podemos clasificar a los algoritmos genético según la forma en la que sus cromosomas son representados, siendo las codificaciones binaria y real las más comunes y utilizadas.

Definimos a un algoritmo genético como binario cuando cada cromosoma se representa como una cada de bits que se compone de 0 y 1. Esto resulta muy útil cuando el problema es discreto o combinatorio como puede ser la selección de variables. Los operadores genéticos se aplican de las siguientes maneras:

* El crossover consiste en intercambios de subcadenas entre padres
* La mutación, por su lado, implica la alteración de un bit de 0 a 1 o viceversa

A pesar de que estos algoritmos son bien simples pero robustos, pueden requerir de cadenas muy largas para lograr un alto nivel de precisión en problemas continuos, incrementando su complejidad.

Por otro lado, un algoritmo genético real representa a los cromosomas directamente como vectores de números ente los valores reales. Esto permite trabajar de manera más natural en problemas de optimización continua como lo es el ajuste de parámetros en redes neuronales. En este caso, los operadores genéticos se adaptan de la siguiente manera:

* El crossover puede implementarse como interpolaciones o combinaciones lineales de los valores de los padres
* La mutación se aplica añadiendo pequeñas perturbaciones aleatorias como el ruido gaussiano

En síntesis, la diferencia esencial entre ambos enfoques radica en la naturaleza de la representación del cromosoma y los operadores asociados. Mientras que los AG binarios son más adecuados para problemas discretos, los AG reales ofrecen ventajas significativas en contextos de optimización numérica y continua, al trabajar directamente en el espacio de búsqueda sin necesidad de mapeos intermedios.

**Casos de aplicación**

* **Selección de variables (Feature Selection) en Machine Learning**

Se aplican AG para seleccionar subconjuntos óptimos de variables predictoras para problemas de clasificación. Esto permitió equilibrar la reducción de dimensionalidad con la mejora en la precisión, siendo esta una de las primeras aplicaciones de AG en modelos de machine learning.

**Fuente:** W. Siedlecki and J. Sklansky. 1989. A note on genetic algorithms for large-scale feature selection. Pattern Recogn. Lett. 10, 5 (November 1989), 335–347. <https://doi.org/10.1016/0167-8655(89)90037-8>

* **Optimización de parámetros en redes neuronales**

En este caso, se utiliza un AG para optimizar los pesos de una red neuronal en lugar de aplicar backpropagation. Este método demuestra las ventajas que existen en evitar mínimos locales y mejorar de este modo la robustez en espacios de búsqueda no convexos.

**Fuente:** D. J. Montana and L. Davis, “Training Feed-Forward Neural Networks Using Genetic Algorithms,” Proceedings of the 11th International Joint Conference on Artificial Intelligence, San Mateo, 20-25 August 1989, pp. 762-767.

* **Optimización del tráfico urbano**

Se aplicaron AG como una estrategia de optimización del tráfico definiendo a cada cromosoma como un conjunto de parámetros de temporización de los semáforos. La función fitness en este caso evalúa el rendimiento de esta configuración en términos de indicadores como el tiempo promedio de espera, densidad vehicular y flujo de la red. A través de los operadores genéticos de selección, cruce y mutación, los AG generan configuraciones cada vez más eficientes, explorando un espacio de soluciones extremadamente grande.

**Fuente:** Halim Ceylan, Michael G.H Bell, Traffic signal timing optimisation based on genetic algorithm approach, including drivers’ routing, Transportation Research Part B: Methodological, Volume 38, Issue 4, 2004, Pages 329-342, ISSN 0191-2615, <https://doi.org/10.1016/S0191-2615(03)00015-8> (<https://www.sciencedirect.com/science/article/pii/S0191261503000158>)

**Implementación open-source**

* **Librería DEAP en Python** es una librería muy conocida, se pueden construir AG de manera modular, incluyendo AG reales, binarios y permutaciones. Permite definir funciones de fitness, cromosomas, operadores de selección, cruce y mutación de acuerdo con el problema que se busque resolver.
* **PyGAD en Python**, otra librería ampliamente utilizada en el área de machine learning y optimización de funciones. Permite la integración con redes neuronales para la optimización de pesos, tiene diferentes tipos de selección y cruce, así como la posibilidad de personalizar la función objetivo.
* **GA en R**, este paquete permite trabajar con AG binarios, reales y permutaciones también, con opciones de personalización en los operadores genéticos y criterios de convergencia.

***Que diferencia existen entre el algoritmo Adaboost y Gradiente Boosting. Explique brevemente. De una explicación intuitiva de cómo trabaja el Gradiente Boosting.***

Tanto el Adaboost como el Gradient Boosting son algoritmos que pertenecen a la familia del boosting, en la cual se combinan modelos débiles de manera secuencial para poder obtener un predictor más preciso.

Por su lado, el Adaboost (cuyo nombre proviene de Adaptative Boosting) ajusta el entrenamiento mediante la reponderación de los ejemplos de entrenamiento. Es decir, en cada iteración los casos que son mal clasificados reciben un peso mayor, de forma que el próximo clasificador se concentre en estos. Esto resulta en un modelo compuesto por una combinación ponderada de los clasificadores débiles, donde cada uno tiene un peso proporcional al desempeño.

Por otro lado, el Gradient Boosting interpreta que el boosting es un problema de optimización en el espacio de funciones. Cada modelo nuevo se construye ara poder aproximar el descenso de gradiente de la función de pérdida, corrigiendo los residuos del modelo anterior. Es decir, en lugar de modificar los pesos de las observaciones como en el Adaboost, se añade de manera stagewise nuevas funciones que minimizar la pérdida elegida.

**Explicación de Gradient Boosting**

De acuerdo con Friedman (2001), el Gradient Boosting puede entenderse como un procedimiento de aproximación aditiva guiado por el gradiente:

1. Se comienza con un modelo simple, por ejemplo, una constante que minimiza la pérdida
2. Se calculan los residuos o pseudorespuestas, que representan la dirección del gradiente negativo de la función de pérdida
3. Se ajusta un nuevo modelo débil (como un árbol pequeño) a dichos residuos
4. El modelo global se actualiza sumando una fracción de este nuevo ajuste
5. El proceso se repite, y cada paso constituye un “boost” que aproxima cada vez más a la función objetivo

En términos más simples, el Gradient Boosting actúa como un proceso de corrección sucesiva de errores donde cada nuevo modelo corrige lo que los anteriores no lograron explicar.

**1.Dar la topología (dibujo y conexiones) y las funciones de activación convenientes para entrenar una red que estime las probabilidades de pertenecer a cada una de tres clases en un problema con 4 variables independientes.**

Para podes estimar las probabilidades de pertenecer a una de 3 clases dado un problema con 4 variables independientes, lo más adecuado sería utilizar una red neuronal de tipo Perceptrón Multicapa (MLP) donde:

* La **capa de entrada** está compuesta por 4 neuronas, una por cada variable independiente
* Haya **1 o 2 capas ocultas** con un número moderado de neuronas con función de activación sea ReLU o tanh, ya que permiten introducir no linealidad y el aprendizaje de relaciones complejas
* La capa de salida serán una neurona por cada una de las 3 clases con función de activación de tipo softmax, la cual convierte las salidas en probabilidades que suman 1
* La función de pérdida es categorical cross-entropy

De este modo, el esquema quedaría algo así

A diagram of a network

AI-generated content may be incorrect.

**¿Qué aporta el modelo Radial Base a las redes Multilayer Perceptron?**

El modelo de Redes de Base Radial (RBF) aporta a los Perceptrones Multicapa (MLP) una alternativa nueva a la manera en que se representa y procesa la información. En una MLP tradicional, las capas ocultas trabajan de forma tal que realizan transformaciones globales mediante combinaciones lineales de los pesos y funciones de activación sigmoides, ya sean tanh o ReLU. De este modo, logran captar relaciones no lineales complejas, lo que requiere de un mayor mantenimiento y ajustes más finos en el número de capas, tasa de aprendizaje e inicialización de los pesos que lo vuelve bastante costoso.

Por otro lado, en un modelo RBF la capa oculta utiliza funciones de base radial. Es decir, que cada neurona responde fuertemente a patrones de entrada cercanos a su centro, por lo que el modelo sea más local (ya que aprende regiones en el espacio de entrada). Para ello, el aprendizaje se divide en dos etapas:

1. Determinar los centros de las funciones radiales, como puede ser un clustering K-means
2. Ajustar los pesos de la salida, donde se suelen usar modelos lineales

De este modo, los RBF aportan a los MLP:

* Mayor interpretabilidad al modelo ya que cada neurona oculta se asocia a una región del espacio de entrada
* Más rapidez de entrenamiento, ya que separa el ajuste de los centros y los pesos
* En problemas con datos bien distribuidos, las RBF capturan relaciones locales mejor que las MLP de forma que aumenta la interpolación
* Más robustez ya que, al enfocarse en vecindades locales, suelen ser menos sensibles al ruido global

**¿Qué es una red neuronal recurrente clásica? ¿Las redes LSTM en que se diferencian de las redes recurrentes clásicas? De un diagrama de su topología y explíquela brevemente. Cantidad de parámetros que utiliza en base a su configuración. Genere un ejemplo completo en Python de aplicación de un red LSTM para predecir una serie de tiempo arbitraria. Pruebe diferentes configuraciones de capas y cantidad de neuronas**

AAAA