Optimización usando técnicas de cómputo de alto rendimiento aplicado a la IA

Optimization using high-performance computing techniques applied to AI

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Índice

1. MPI

2. Aprendizaje por refuerzo

Reinforcement Learning

Se basa en experiencias y simulaciones, con prueba y error, recibiendo recompensas con las acciones tomadas (también pueden ser negativas). Nadie le dice al agente que hacer, toma las decisiones con diferentes estrategias. En la etapa de entrenamiento suele ser de forma aleatoria. Una vez tiene un feedback del entrenamiento se toma las decisiones maximizando las recompensas obtenidas en experiencias pasadas.

Algoritmo Q-Learning: Mezcla entre programación dinámica y Monte Carlo.

R=Matriz de recompensas.

Q=Matriz (Estados x Acciones). Que acción elegir en cada estado. (mayor valor)

Aprende el camino si es una buena acción, Back Propagation (Como en redes neuronales).

S=estado actual. A=acción tomada. S'=Estado siguiente. A_i=una acción.

$$\mathbf{Q(S, A) = (1-\alpha)*Q(S, A) + \alpha*(R(S, A) + \gamma*max_i\ Q(S', A_i))}$$

Hiperparametros:

• *α (tasa de aprendizaje):*

Debería disminuir a medida que continúa adquiriendo una base de conocimientos cada vez mayor.

• *γ (factor de descuento):*

A medida que se acerca cada vez más al valor límite, su preferencia por la recompensa a corto plazo debería aumentar, ya que no estará el tiempo suficiente para obtener la recompensa a largo plazo, lo que significa que su gamma debería disminuir.

• *ϵ:* Evita que la acción siga siempre la misma ruta.

A medida que desarrollamos nuestra estrategia, tenemos menos necesidad de exploración y más explotación para obtener más utilidad de nuestra política, por lo que en vez de utilizar un valor fijo, a medida que aumentan los ensayos, épsilon debería disminuir. Al principio un épsilon alto genera más episodios de exploración y al final un épsilon bajo explota el conocimiento aprendido.

Mejorar:

Paralelización del entorno:

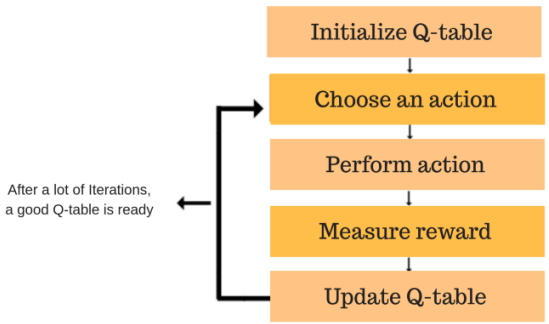
- Ejecutar en varios “workers“ el programa en la misma celda.
- Ejecutar en varios “workers” el programa en diferentes celdas.
- Ejecutar varios “workers” asignando secciones del mapa.
- Recorrer la matriz Q e ir actualizando los valores. Varios workers toda la matriz, dividir la matriz.

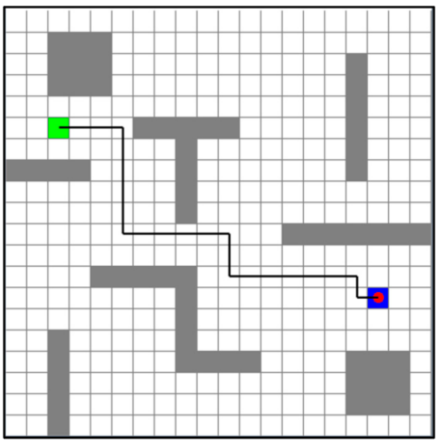
Optimizar los hiperparámetros:

- Ejecutar en varios “workers” el mismo programa, pero con diferentes hiperparámetros

Estrategias de exploración:

DQN (Deep Q-Network): tipo de algoritmo de aprendizaje por refuerzo que combina Deep Learning con Q-Learning.





(a)

https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/

https://www.mdpi.com/2073-8994/13/6/1057

3. Programación Evolutiva

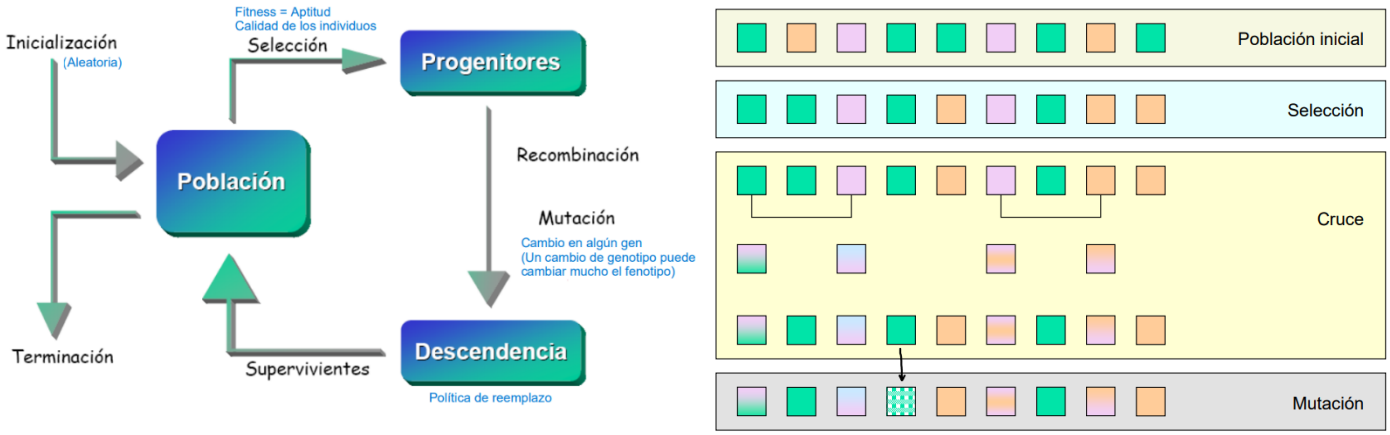
La programación evolutiva es una técnica de optimización inspirada en la teoría de la evolución biológica. Se basa en el concepto de selección natural y evolución de las poblaciones para encontrar soluciones a problemas complejos.

Una población está compuesta por individuos. Un individuo tiene un cromosoma, que tiene uno o varios genes, que a su vez cada gen tiene 1 o varios alelos. Los individuos se pueden representar:

- Binarios: Cada alelo es un bit. Los rangos de números naturales en binario son de 2^N, para codificar un 127 en binario se necesitan 2^7 bits. Y si queremos añadir números reales con una cierta precisión este número de bits aumenta.
- Reales: Números reales, este es más fácil de manejar.

El fenotipo (Decodificación) de un individuo son los rasgos observables, es decir el valor numérico.

El genotipo (Codificación) es la composición genética de un individuo.



Inicialización: Hay que inicializar los individuos de la población.

Tipos: Aleatoria, Elección de población inicial.

Métodos de selección: Como elegir a los individuos de la población que se van a cruzar y mutar.

Tipos: Ruleta, torneo, estocástico, truncamiento, restos. Otros menos conocidos, Boltzmann, roulette-wheel selection (parecido a ruleta, pero es más optimizable), exponential ranking selection.

Métodos de cruce: Individuos de la población que van a cruzarse, tiene una probabilidad, suele ser 0.6.

Tipos: Un Punto, Dos Puntos, Uniforme, Aritmetico, SBX, PMX, BLX, Mask-based crossover, Order crossover. Otros menos conocidos. Cycle crossover, Position crossover, Tree-based crossover.

Métodos de mutación: Individuos de la población que una vez cruzados (o no) van a ser mutados, con una población (Suele ser baja en binarios 0.05 reales 0.3)

Tipos: Único, Múltiple, Uniforme, Intercambio, Inserción, Desplazamiento, No uniforme, Duplicación, Adaptativo.

Elitismo: Asignar una parte de la población a los mejores, así garantizando la supervivencia de los más aptos.

Modificar aptitudes/Desplazamiento: Para maximización y minimización. Devuelve valores positivos.

Convertir problemas de maximización a minimización y viceversa.

Presión Selectiva: La mayor o menor tendencia a favorecer a los individuos más aptos.

Bajar, aumentar o mantener.

Escalado de la adaptación: Permite controlar la diversidad de las aptitudes. Tipos: Lineal, Sigma, Potencial, basado en orden.

Codificación Gray: En muchos problemas de optimización numérica se ha comprobado la utilidad de utilizar un cromosoma representado como un vector de números reales

Criterios de terminación: Búsqueda del mejor criterio de terminación para un problema.

Tipos: Generaciones, llegar al optimo

Mutación variable

Tasa de mutación variable que se vaya reduciendo a medida que avanza la evolución [FOG89].

Modificando los parámetros de entrada:

Tamaño de población, numero de generaciones, probabilidad de cruce, mutación.

Tamaño de individuo, precisión.

Mejorar:

Se puede mejorar todo lo anterior

4. Aprendizaje No Supervisado

Distancia entre Individuos: Manhattan, Euclídea o Chebychev. Distancia Clusters: Centroide, Enlace simple o Enlace completo.

Elección del número de clusters.

Regiones Voronoi.

Índices que miden lo compacto de la solución, Dunn, Davies-Bouldin, Coeficiente de Silueta.

Algoritmos de clustering jerárquico aglomerativo

- FASE 1: Crear la matriz de distancias inicial D

Es una matriz simétrica (basta con usar una de las matrices triangulares) -

- FASE 2: Agrupación de Individuos
    - Partición inicial P_0: Cada objeto es un cluster
    - Calcular la partición siguiente usando la matriz de distancias D
        - Elegir los dos clusters más cercanos. Serán la fila y la columna del mínimo de la matriz D
        - Agrupar los dos en un cluster. Eliminar de la matriz la fila y columna de los clusters agrupados. Generar la nueva matriz de distancias D.
            - Añadir una fila y una columna con el cluster nuevo
            - Calcular la distancia del resto de clusters al cluster nuevo
    - Repetir paso 2 hasta tener sólo un cluster con todos los individuos
      • Representar el dendograma (árbol de clasificación)

La complejidad con los enlaces simple y completo tienen un coste cúbico O(n3). Existen implementaciones más eficientes ($n2$).

Algoritmos de clustering basados en particiones - Algoritmo de K-Medias:

Se fija un valor **k**.

1. Inicializar los k centros (o centroides) de los clusters de forma aleatoria.

- Generando puntos aleatorios en el espacio dimensional

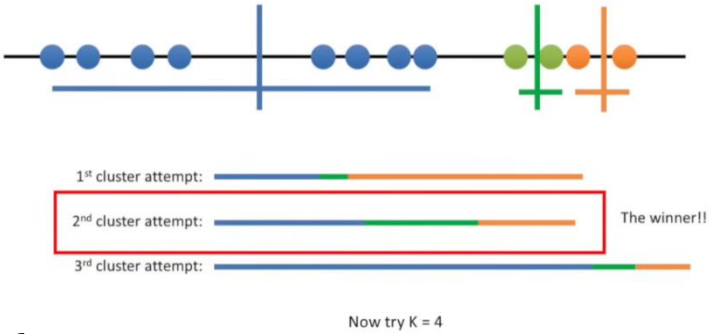
- Seleccionando aleatoriamente individuos

2. Repite el siguiente proceso hasta que los centros no cambien.

1. Fase de asignación: Para cada individuo se le asigna el cluster más cercano. Requiere el uso de una **distancia** (normalmente euclídea, también se puede usar manhattan o Chebychev)

2. Actualiza el centro de los clusters. Se calcula con la media de los individuos del

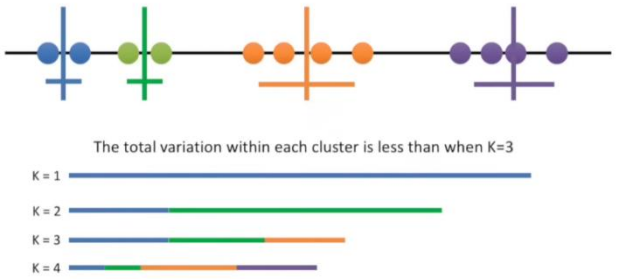
Como los clusters se generan aleatoriamente, no siempre va a dar un buen resultado a la primera. Por lo que se pasa por parámetro cuantas veces **se repite hasta encontrar el mejor.**



Encontrar el **valor ideal para k.**

Se pueden comparar con la variación total. La variación es el espacio de cada cluster de los más lejanos.

Se puede comprobar el mejor valor con un diagrama de codo.



Puede haber **varias dimensiones** por lo que el cálculo de la distancia se complica.

distancia euclídea = raizcuadrada(x0^2 + x1^2 + ... + xn^2)

Mejorar:

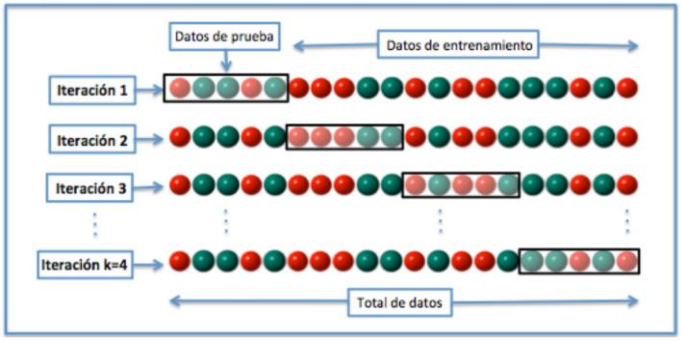
- Cálculo de distancias de los individuos:
  - Asignar conjunto de individuos
  - Asignar de individuo en individuo
  - Asignar el cálculo de las distancias de un individuo a un conjunto de clusters, si hay muchos.
  - Comprobar la eficiencia con muchas dimensiones
- Proceso de búsqueda:
  - Mejor configuración para cada numero de clusters.
  - Valor ideal para k:

5. Aprendizaje Supervisado

- Preprocesar datos e inicialización
- Fase de aprendizaje/entrenamiento
- Evaluación

Validación Cruzada *(También para supervisado):*

- 1. Los datos se dividen aleatoriamente en k subconjuntos iguales
- 2. Se entrena el conjunto con (k-1) y se valida con el restante
- 3. Se repite k veces el paso 2, cambiando el conjunto que se usa para validar
- 4. Como medida de error final se suele presentar la media de las k medidas de error de validación (aunque puede ser interesante comprobar que las k medidas sean similares)



KNN

Es simple pero potente y se basa en la idea de que los puntos de datos similares tienden a agruparse en el espacio de características.

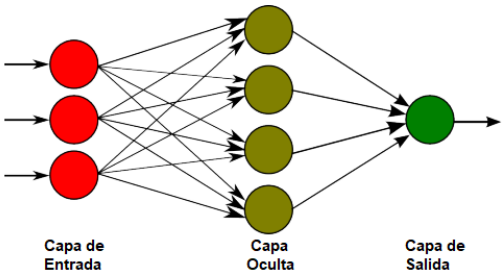
1. Empezar con un dataset con categorías conocidas.
2. Añadir un nuevo individuo con categoría desconocida.
3. Clasificar el individuo con los k vecinos más cercanos.

Se puede aplicar a mapas de calor

No hay una forma de determinar el mejor valor para k, por lo que hay que probar con varias ejecuciones. valores pequeños de k crea sonido. Valores grandes con pocos datos hará que siempre sea la misma categoría

Redes Neuronales

- La primera columna es la capa de entrada de la variable.
- La última columna es la capa de salida con todas las posibles salidas.
- Las columnas intermedias son las capas ocultas



Cada **neurona** hace una operación simple. **Suma los valores de todas las neuronas de la columna anterior**. Estos valores multiplicados por un peso que determina la importancia de conexión entre las dos neuronas. Todas las neuronas conectadas tienen un peso que irán cambiando durante el proceso de aprendizaje. Además, un valor bias puede ser añado al valor calculado. No es un valor que viene de una neurona especifica y se escoge antes de la fase de aprendizaje. Puede ser útil para la red. Con el valor calculado se aplica a una función de activación, para obtener el valor final. Se suele usar para dar un valor entre [0-1].

Proceso de aprendizaje/entrenamiento:

Lo que queremos que haga la red neuronal, es que dado una entrada devuelva una salida. Al principio no va a ser así, solo por suerte devuelve la salida correcta. Por esto se genera la etapa de aprendizaje, en la que cada entrada tiene asociada una etiqueta, para explicar que salida debería de haber adivinado.

- Acierta: los parámetros actuales se guardan, y se envía la siguiente entrada
- Falla: los pesos son cambiados. Se suele usar **backpropagation**

Mejorar:

- Paralelizar:
  - Inicialización, dividir la carga a la hora de crear la red.
  - Entrenamiento, Se puede enviar un conjunto de individuos a varios "workers" para que vaya aprendiendo. Propagación hacia delante y hacia atrás en redes neuronales o vecinos más cercanos.
- Optimizar parámetros: Encontrar el mejor valor para la tasa de aprendizaje o el valor k.
-