Universidad Nacional de Colombia - Sede Bogotá
Facultad de Ingeniería

Inteligencia Artificial y Minirobots **I semestre 2026**
Código: 2023251
Créditos: 3

ORIENTACIÓN
El curso está dirigido a estudiantes de ingenierías mecatrónica, mecánica, eléctrica, electrónica, industrial, y de sistemas que estén interesados en obtener y/o ampliar sus perspectivas de fundamentación en técnicas básicas de inteligencia artificial, con aplicaciones a problemas de ingeniería.

OBJETIVOS

- Conocer diferentes técnicas de inteligencia artificial, así como su aplicación en problemas de ingeniería.

CONTENIDO

Temas del profesor José Martínez P.:
1. Panorama de la IA. Un acercamiento a la IA.
    Enfoques para afrontar la IA.
    Sistemas. Modelo generalizado. Sistemas simples, complicados, caóticamente desorganizados, complejos. Complejidad computacional. Herramientas de IA.
2. Autómatas Celulares.
    Vecindades, definición formal, propiedades esenciales.
    Autómatas celulares de 1D.
    Autómatas celulares 2D, Life.
    Aplicaciones de los AC.

3. Algoritmos Genéticos. Se aplica para encontrar problemas de alta complejidad computacional.
    Introducción, descripción del A. G. simple.
    Solución de problemas
    Solución del problema de asignación.
    Aplicaciones.

4. Programación Genética. Se aplica para hacer programación óptima de máquinas.

    Conceptos básicos.
    Conjunto de terminales, conjunto de funciones, función de aptitud.
    Operadores.
    Solución de problemas por PG.
    Aplicaciones.

5. Redes Neuronales. .
    Introducción.

Neurona natural, neurona artificial, función de activación.
Redes Feed-forward multicapa. Algoritmo de retro-propagación,
Conceptos de aprendizaje profundo.
Aplicaciones.

6. Introducción a ML

Introducción
¿Qué es ML? Porque el uso de ML Aplicaciones.
Tipos de aprendizaje, aprendizaje supervisado, semi-supervisado, no-supervisado y reforzado. Ejemplos.
Proyectos de ML.
ML oscura

7. Inteligencia Artificial Generativa

Introducción
IA Generativa.LLMs (Large Language Models), Transformers, Modelos Fundacionales. .
Comportamiento de las herramientas de IA Generativa
Inquietudes sobre la IA generativa. Ejercicios

Temas del profesor Gustavo Pérez H.:

8. Sistemas y Redes de lógica fuzzy (difusa). Se puede aplicar a muchos campos de la robótica y automatización.

Conjuntos clásicos y lógica clásica.
Conjuntos difusos, operaciones, cálculo proposicional difuso, lógica difusa, Aplicaciones con Sistemas Difusos
Aplicaciones con Redes Difusas
Aprendizaje de Máquina. Aplicaciones.

Temas del profesor Flavio Prieto:

9. Percepción Visual en Robótica

Principios de visión por computador en robótica.
Extracción de características (puntos clave, bordes, texturas).
Uso de CNNs para detección y reconocimiento de objetos en robots.

10. Fundamentos del Control Servovisual

Definición y clasificación: control basado en posición vs control basado en imagen.
Ecuaciones de control y relación entre imagen y movimiento.
Estimación de pose y cinemática inversa con visión.

11. Aprendizaje Profundo en Servovisualización

Uso de redes profundas para estimación de pose y puntos de control visual.
Redes convolucionales (CNNs) aplicadas a visión robótica.
Redes recurrentes (RNNs/LSTMs) para seguimiento de trayectorias visuales.

12. Aprendizaje por Refuerzo en Control Visual

Introducción al Aprendizaje por Refuerzo (RL) en robótica.
Deep Reinforcement Learning aplicado a control autónomo con visión.
Ejemplos: navegación de minirobots, seguimiento de objetos, evitación de obstáculos.

13. Aplicaciones Prácticas en Minirobots

Seguimiento visual de un objeto móvil.
Control visual para manipulación (ej. agarre de objetos).
Navegación autónoma basada en visión.

METODOLOGÍA

Clases magistrales (60%).
Talleres y ejercicios aplicados de programación (40%).
Lecturas y ejercicios complementarios (extra-clase).

EVALUACIÓN

40% Proyectos (Presentación oral y escrita).
35% Talleres y tareas.
25% Examen.

Profesor José Martínez 33.33…%
Profesor Flavio Prieto 33.33…%
Profesor Gustavo Pérez 33.33…%

BIBLIOGRAFÍA

1. Winston, P. H. ***Artificial Intelligence***, Third Edition. Addison-Wesley, Reading, Massachusetts, 1992.
2. Chang, Chin-Liang y Lee, Richard Char-Tung. ***Symbolic Logic and Mechanical Theorem Proving***. Academic Press, New York-San Francisco-London, 1973.
3. Nilsson, J. Nils. ***Principles of Artificial Intelligence***. Springer-Verlag, Berlin-Heidelberg-New York, 1982.
4. Russell, S. y Norvig, P. ***Artificial Intelligence, A Modern Approach***. Prentice-Hall, Upper Saddle River, New Jersey, 1995.
5. Kusiak, A. ***Intelligent Manufacturing Systems.*** Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
6. Wright, P. K. y Bourne, D. A. ***Manufacturing Intelligence***. Addison-Wesley, Reading, Massachusetts, 1988.
7. Horn, B. K. P. ***Robot Vision***. The MIT Press, Cambridge, Massachusetts, 1988.
8. Dougherty, E. R. y Giardina, Ch. R. ***Mathematical Methods for Artificial Intelligence and Autonomous Systems***. Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

9. Klir, G. J. y Folger, T. A. ***Fuzzy Sets, Uncertainty, and Information***. Prentice-Hall, Englewood Cliffs, New Jersey, 1988.
10. Klir, G. J. y Yuan, Bo ***Fuzzy Sets and Fuzzy Logic***: *Theory and Applications*. Prentice-Hall, Englewood Cliffs, New Jersey, 1995
11. Waterman, Donald A. ***A Guide to Expert Systems***. Addison-Wesley, Reading, Massachusetts, 1986.
12. Jaime Malpica A., ***Notas de clase***.
13. José Martínez P., ***Notas de clase***.
14. Gustavo Pérez H., ***Notas de clase***.

1. Panorama de la IA
José J. Martínez P.

Jmcursosingenieria111@gmail.com
Febrero 2026

Un acercamiento a la IA

La IA es la tecnología más fascinante y revolucionaria de nuestro tiempo, pues impacta cada vez más la sociedad mundial, transformando la gran mayoría de quehaceres de la humanidad. Se ha convertido en el nuevo paradigma de la ingeniería, pues en este momento no se puede concebir un nuevo producto o servicio que no incluya alguna de sus herramientas ya sea en su diseño, en su proceso de construcción o en su operación. Desde 2023 la IA ha tenido más titulares periodísticos que cualquier otra tecnología y que muchos otros temas.

Una primera definición es la de IA General y de IA Estrecha. La IA General busca que todo un gran sistema funcione con IA, es decir la automatización completa, por ejemplo, de una ciudad usando IA. Esta todavía es una meta a largo plazo y muchos autores consideran imposible de lograrla, aunque si bien ya hay algunos proyectos que se han comenzado a construir como el proyecto Arup en el Valley Cloud en el sur de China, se considera que todavía falta mucha interoperabilidad. De manera que lo que se trabaja actualmente es la IA estrecha, el uso de IA para la solución de problemas aislados, específicos.

Los desarrollos recientes especialmente en Machine Learning, ML, Deep Learning, DL, e IA Generativa, IAGen, especialmente esta última, han llevado directamente a que la sociedad cambie a nivel planetario, por las formas en que ahora las personas perciben el mundo. Aplicaciones de IA Generativa como Chat-GPT, DeepSeek, y otras, confrontan el sistema de educación, el sistema de justicia y el ejercicio profesional en muchas especialidades, poniendo en riesgo muchos puestos de trabajo; y si somos audaces también creando muchos puestos de trabajo. Vivimos actualmente momentos de verdadera revolución por la disponibilidad y utilización de estas herramientas, y ahora su utilización como Apis en el desarrollo de nuevas aplicaciones. Volviendo a la IA General, algunos autores consideran que la IAGen es uno de sus componentes claves, por la posibilidad de generar contenido, simular el pensamiento humano y por la posibilidad de tomar decisiones complejas, por ejemplo en sistemas de producción, que algunos autores consideran se puede lograr relativamente pronto.

Además, la IA y la IAGen se ha comenzado a utilizar en otros dominios, como en el caso de las máquinas donde se pretende dotarlas con razonamiento cognitivo, es decir mejorar su forma de procesar la información para solucionar problemas, tomar decisiones, llegar a conclusiones, tener conciencia espacial y actuar de manera significativa en contextos del mundo real. Lo que se conoce como IA Física.

Considerando el potencial transformador de la IA, diversos gobiernos y organizaciones consideran esta tecnología como un elemento importante en el juego geopolítico que se está presentado a nivel global en este momento, como es el caso de los chips H20 de NVIDIA. Por lo que los países

avanzados invierten ingentes recursos en muchos planes estratégicos para fomentar la innovación con la utilización y desarrollo de herramientas de IA.

Los inicios de la IA se dieron con la construcción del primer computador americano en 1943, se concretaron las posibilidades de implementar procesos de razonamiento humano, lo que inspiró a varios científicos a discutir la posibilidad de llegar a un "cerebro electrónico". El término Inteligencia Artificial fue acuñado en 1956, en un taller realizado en Darmouth College, al que asistieron muchos de los que posteriormente fueron los líderes en investigación en este campo, como H. Simon, J. McCarthy, C. Shannon, A Newell y M. Minsky. El documento que se produjo en esta reunión define "***el problema de la inteligencia artificial es el de construir una máquina, que, si se comporta de manera similar al comportamiento que realice un ser humano, esta sería llamada inteligente***". Aunque han aparecido diversas opiniones sobre la definición de la IA, aquí seguimos esta orientación anterior. Veamos ahora los cinco diferentes enfoques que se han trabajado para llevar al computador esos procesos de toma de decisiones humanas.

1. Enfoques para afrontar la IA

1.1 Pensar Humanamente

Si decimos que un algoritmo "piensa" como un humano, debemos tener alguna forma de determinar cómo piensa un humano. Así que se necesita entrar en el trabajo real de cómo funciona la mente humana. La mejor forma de hacerlo es a través de experimentación psicológica. Una vez se tenga una teoría precisa y suficiente de la mente, se vuelve posible expresar la teoría como un programa de computador. Si la entrada/salida y el comportamiento en el tiempo concuerdan con el comportamiento humano, esta es una evidencia de que algunos mecanismos del programa pueden operar en los humanos. El campo interdisciplinario de la ciencia cognitiva reúne modelos computacionales a partir de la IA y de técnicas experimentales de la psicología para tratar de construir teorías precisas y probables del trabajo de la mente humana.

1.2 Inteligencia Artificial Simbólica

Como vimos, Aristóteles fue el primero en tratar de codificar el "pensamiento correcto", esto es, procesos de razonamiento irrefutables. Sus famosos silogismos proveen patrones de estructuras de argumentos, que siempre llevan a conclusiones correctas a partir de premisas correctas.

En la conferencia de 1956 en Darmouth College se planteó que "cada aspecto del aprendizaje o cualquier otra característica de la inteligencia puede, en principio, describirse en forma tan precisa (formal) que se pueda construir una máquina para simularla. Se tratará de encontrar cómo hacer que las máquinas utilicen el lenguaje, formen abstracciones y conceptos, solucionen la clase de problemas ahora reservados para los humanos y se mejoren a sí mismas[i]". Esta definición se conoce como IA simbólica.

Sin embargo, este enfoque tiene un obstáculo importante, en la gran generalidad de casos, no es fácil a partir de conocimiento informal, establecer ese conocimiento en términos formales, en notación lógica. Este enfoque solo se utiliza en ambientes completamente controlados.

1.3 El enfoque de agentes

En este enfoque, la IA se ve como el estudio y construcción de agentes inteligentes. En el enfoque de IA simbólica, todo el énfasis está en las inferencias correctas. Un agente inteligente puede hacer inferencias correctas, debido a que una forma de actuar racionalmente es razonar lógicamente para llegar a la conclusión, de que dada una acción se llegará a las metas que uno ha definido, y luego actuar sobre esa conclusión.

De la figura 1.1, se puede observar que un agente inteligente es un ente que percibe y actúa. El ambiente produce mucha información, que el agente percibe y que filtra la información pertinente a través de sus sensores, que llega a su modelo mental, que la analiza y decide ordenar a los actuadores ejecutar acciones.

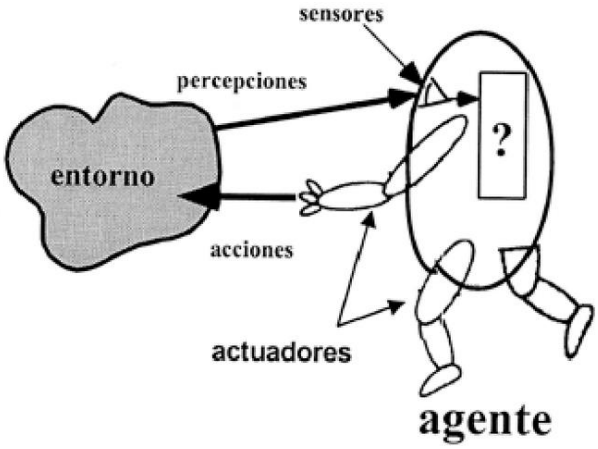


Figura 1.1. Un agente inteligente.

Algo interesante de observar también en la figura 1.1 es la connotación humanoide del agente. Algo que de alguna manera lo podemos asumir al vernos como agentes inteligentes: con base en nuestro modelo interno, recibimos la información que nos provee el medio ambiente, filtrada por nuestros sensores, la procesamos y la interpretamos, con base en nuestro modelo interno, para tomar nuestras decisiones, en espera de obtener algún tipo de recompensa.

1.4 Sistemas bio-inspirados

Cuando observamos los seres vivos, encontramos que existe una gran diversidad de especies, que aprovechan diversos nichos ecológicos para mantenerse y reproducirse. Cada ser es autónomo, capaz de auto-organizarse y crear estrategias, para mantenerse vivo y lograr enviar su información genética al futuro. Su lucha permanente en un entorno incierto, la generación de estrategias para atrapar presas o para eludir predadores, son todas acciones “inteligentes”. La observación de estos comportamientos y de sus procesos, para la solución de los problemas que enfrentan, permite pensar en el desarrollo de herramientas de IA.

1.5 El enfoque de datos

Los datos, provienen de hechos reales, irrefutables, que de una u otra manera han sucedido, son verdades, son la experiencia humana en términos digitales. que podemos usarlos para solucionar problemas complejos, ¿que solo soluciona el humano? Esto ha llevado al desarrollo de nuevas ciencias y tecnologías como la Ciencia de Datos, el Big Data e incluso la Física Social.

Con esta disponibilidad de datos, experiencias de miles de humanos, a través de su estudio, procesamiento y análisis, no solamente se construyen modelos predictivos de sistemas bastante complejos, sino también se obtienen ideas útiles, es lo que se conoce como Machine Learning, ML. A partir de ahí, se ha seguido con Deep Learning y IAGen. Este enfoque de datos es tan importante en este momento, que en algunos ambientes se considera que solo esto es IA, cuando realmente es solo un subcampo de la IA.

3. Pero ¿qué es inteligencia?

Entre las diferentes definiciones de inteligencia se pueden resaltar algunas frases: mayor importancia para la vida práctica; aprender o sacar provecho de la experiencia; adaptarse adecuadamente; aprendió o puede aprender; resolver problemas nuevos; superar obstáculos; adaptarse y prosperar en cualquier entorno dado. Estas frases conllevan metas, aprendizaje, memoria, adaptación e incertidumbre sobre el ambiente.

Sin embargo, no aparece la variable tiempo, básica a la "inteligencia". Por lo que incluimos un sinónimo como "rápidamente" y decir: adaptarse adecuadamente rápidamente; ha aprendido o puede aprender rápidamente; resolver problemas nuevos rápidamente; adaptarse y prosperar en cualquier entorno rápidamente. Aunque es claro que en algunos casos no funcione tan bien.

Por otra parte, una de las estrategias de la biología, para lograr la creación del individuo, ha sido la de proveer miles de soluciones al problema de la incertidumbre del ambiente. De esta manera, algunas de esos miles de soluciones, un muestreo de la explosión exponencial de soluciones podrá interactuar exitosamente con ese ambiente y enviar su información genética al futuro, mientras las otras simplemente fracasarán en el proceso.

4. Sistemas

Con el fin de **predecir** el comportamiento de los sistemas en el tiempo, vamos a definir sistema, luego un modelo generalizado de sistema y con base en ese modelo clasificar los sistemas para predecir su comportamiento en el tiempo.

Un **sistema** es un conjunto de elementos, componentes, subsistemas o agentes; que están ligados a través de enlaces, vecindades, canales; de alguna manera. Esta ligazón genera un comportamiento nuevo, **emergente**, que generalmente no es propio de ninguno de los elementos o agentes que componen el sistema.

4.1 Modelo de sistema matemático generalizado

Veamos un modelo matemático generalizado de lo que es un sistema, considerando sus agentes, sus relaciones y cómo se relaciona el comportamiento de los agentes con el comportamiento global del sistema, con el fin de predecir su comportamiento. Como modelo abstracto es aplicable a cualquier sistema.

Un sistema S se puede definir como el par $S = \{A, R\}$, donde A es el conjunto de agentes que conforman el sistema y R es el conjunto de relaciones o enlaces existentes entre los agentes A del sistema S.

Se define Q como alguna medida del comportamiento emergente del sistema S, a partir de los comportamientos q de cada uno de los componentes A del sistema S, que expresamos como[1]:

$$Q = f(q_1, q_2, \dots, q_n)$$

Donde q_i es el comportamiento del agente a_i para $i = 1, .., n$

Ahora para encontrar el comportamiento dinámico del sistema S en el tiempo, la predicción del comportamiento S en función del cambio de comportamiento de cada uno de los agentes, hacemos:

$$\frac{dQ}{dt} = \frac{df}{dq_1}\frac{dq_1}{dt} + \frac{df}{dq_2}\frac{dq_2}{dt} + \dots + \frac{df}{dq_n}\frac{dq_n}{dt}$$

Por otro lado,

$$\frac{dq_1}{dt} = f'(q_1, q_2, \dots, q_n)$$

$$\frac{dq_2}{dt} = f'(q_1, q_2, \dots, q_n)$$

…

$$\frac{dq_n}{dt} = f'(q_1, q_2, \dots, q_n)$$

Es claro que el resultado de la derivada es diferente para el cambio de comportamiento de cada agente. Entonces ahora, vamos a clasificar ***todos*** los sistemas a partir de estas ecuaciones, en: Sistemas Simples, Sistemas Complicados, Sistemas Caóticamente Desorganizados y Sistemas Complejos.

Sistemas Simples

En los sistemas simples, $S_S = \{A_S, R_S\}$, el número de agentes A_S es muy bajo, del orden de 2, 3 o 4 agentes y el conjunto de relaciones entre esos agentes R_S está definido formalmente, matemáticamente. Además, se conoce el comportamiento de cada agente. Luego no hay ningún problema en encontrar cambios o hacer predicciones en el sistema cuando cambia alguno de los agentes. Por ejemplo, todos los sistemas eléctricos, están compuestos de tres agentes fundamentales: resistencias, condensadores y bobinas. Veamos la ecuación que incluye estos tres elementos en una malla figura 3.1, que tiene una fuente (t):

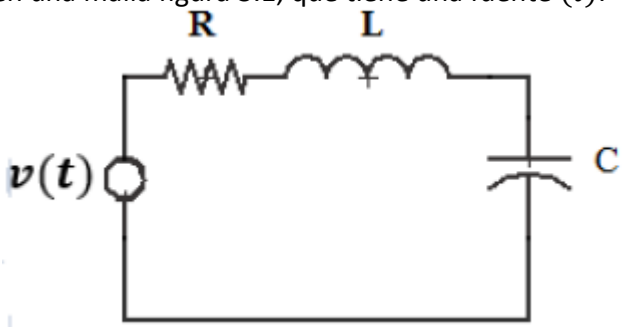


Figura 1.4. Circuito eléctrico con los tres agentes universales.

[1] Ecuación 3.1, página 56, General Systems Theory, L. von Bertalanffy, 1968

$$L\,\frac{di}{dt} + Ri + \frac{1}{C}\int_{t_0}^{t} i\,dt + \; v_c(t_0) = v(t)$$

Donde $\boldsymbol{R}$ es la resistencia, $\boldsymbol{L}$ es la inductancia de la bobina, $\boldsymbol{C}$ la capacitancia del condensador e $\boldsymbol{i}$ es la corriente que fluye por el circuito. $(\boldsymbol{t_0})$ es la carga del condensador en tiempo cero. Donde sus agentes son $\boldsymbol{R}$, $\boldsymbol{L}\ y\ \boldsymbol{C}$, con todos sus operadores lineales como son la multiplicación, la derivada y la integral. Los sistemas que se estudian en diferentes ramas de la ingeniería son sistemas lineales, a los que se ha llegado a través de diferentes abstracciones. Así en el caso de los sistemas mecánicos, muchas de sus ecuaciones son iguales a las de los circuitos eléctricos, esta es la razón de los computadores analógicos. De esta manera, tenemos que, para este tipo de sistemas, las analogías permiten el estudio de diferentes sistemas sin importar su composición física.

Sistemas Complicados

En los sistemas complicados, $S_c = \{A_c,\ R_c\}$, el número de agentes A_c no es muy alto; y el conjunto de relaciones R_c entre los comportamientos de los agentes es $= \{\emptyset\}$. Además de que el comportamiento de cada agente es independiente, también es constante. En otras palabras:

$$\frac{dq_1}{dt} = f'(q_1)$$

$$\frac{dq_2}{dt} = f'(q_2)$$

…

$$\frac{dq_n}{dt} = f'(q_n)$$

De manera que:

$$\frac{dQ}{dt} = \frac{df}{dq_1}f'(q_1) \; + \frac{df}{dq_2}f'(q_2) + \cdots + \frac{df}{dq_n}f'(q_n)$$

El comportamiento dinámico del sistema depende del comportamiento de cada agente individual. Básicamente son los sistemas tecnológicos, construidos por el hombre a través de procesos de ingeniería, siguiendo ciertos procedimientos y diseñados para un ambiente determinado. Aunque su comportamiento no es lineal, es completamente predecible.

Por ejemplo, un carro. Está diseñado para que en el ambiente haya oxígeno disponible para que con el combustible explote en los cilindros del motor. Además, requiere una estructura de carreteras sobre las cuales se desplace con seguridad, siguiendo un reglamento de tránsito, a unas determinadas velocidades, llevando un determinado número de personas. El consumo de combustible es casi proporcional a los recorridos que realiza. Además, cuando un carro se daña, se siguen procedimientos muy precisos, complicados, pero que a la larga permiten su arreglo.

Figura 1.6. Sistemas complicados.

Es interesante observar que estas piezas se modelan en sistemas CAD, que obedecen a una serie de ecuaciones matemáticas. Son el producto de sistemas simples.

Sistemas Caóticamente Desorganizados

En los sistemas caóticamente desorganizados, $S_{cd} = \{A_{cd}, R_{cd}\}$, el número de agentes A_{cd} es alto o muy alto. El conjunto de relaciones R_{cd} entre los comportamientos de los agentes es $= \{\emptyset\}$. En estos sistemas cada agente se comporta de manera **independiente** de los otros agentes y su espacio de estados es el mismo. En momentos diferentes un agente que estaba en un estado puede cambiar a otro estado independientemente, de manera que, para este caso podemos suponer que sus comportamientos son similares, $f'(q_i) \ = \ f'(q)$.

$$\frac{dq_1}{dt} = f'(q)$$

$$\frac{dq_2}{dt} = f'(q)$$

…

$$\frac{dq_n}{dt} = f'(q)$$

De manera que:

$$\frac{dQ}{dt} = \frac{df}{dq_1} f'(q) \ + \frac{df}{dq_2} f'(q) + \cdots + \frac{df}{dq_n} f'(q)$$

Entonces

$$\frac{dQ}{dt} = n \ \frac{df}{dq} f'(q)$$

Se ve aquí que el comportamiento de los sistemas caóticamente desorganizado es la sumatoria de los comportamientos de sus agentes, por eso para el estudio y predicción del comportamiento de estos sistemas se aplica toda la teoría estadística.

Figura 1.7. Sistemas caóticamente desorganizados

Sistemas Complejos

En los sistemas complejos, $S_{co} = \{A_{co}, R_{co}\}$, el número de agentes A_{co} es relativamente alto. El conjunto de relaciones R_c entre los comportamientos de los agentes no se puede plantear en términos formales dado que el comportamiento de un agente **influye** de diversas maneras en el comportamiento de otros agentes. Así que si tomamos, el cambio de comportamiento para el agente q_1:

$$\frac{dq_1}{dt} = f'(q_1,\, q_2, \dots,\, q_n)$$

Derivando

$$\frac{dq_1}{dt} = \frac{\partial f''}{\partial^2 q_1}\frac{\partial^2 q_1}{dt} + \frac{\partial f''}{\partial^2 q_2}\frac{\partial^2 q_2}{dt} + \cdots + \frac{\partial f''}{\partial^2 q_n}\frac{\partial^2 q_n}{dt}$$

Ahora,

$$\frac{\partial^2 q_1}{dt} = \frac{\partial f'''}{\partial^3 q_1}\frac{\partial^3 q_1}{dt} + \frac{\partial f''}{\partial^3 q_2}\frac{\partial^3 q_2}{dt} + \cdots + \frac{\partial f''}{\partial^3 q_n}\frac{\partial^3 q_n}{dt}$$

Podemos seguir derivando, no sabemos hasta donde, para tener el cambio de comportamiento de solo el agente q_1 del sistema. Lo mismo se debe considerar para el resto de los agentes del sistema. Y una vez tengamos todos esos comportamientos ver cómo se obtiene el comportamiento dinámico de todo el sistema. Dado que no podemos conocer estos comportamientos, tampoco podemos conocer el comportamiento global del sistema, en otras palabras desde esta perspectiva, **no podemos predecir** el comportamiento de un sistema complejo.

Complejidad computacional. Otro tipo de complejidad es la ***complejidad computacional*** que tiene que ver con el costo computacional, en términos de tiempo de ejecución, de un algoritmo. Casi que la totalidad de algoritmos que se ejecutan actualmente en los computadores tiene un tiempo de ejecución proporcional al número de transacciones o datos que procesan, es una relación lineal. Sin embargo, hay algoritmos cuyo tiempo de ejecución crece exponencialmente o más, con el número de transacciones o datos que procesan. Por ejemplo, en el Sudoku, el crecimiento es impresionante. Un sudoku de tamaño 2x2 tiene 288 diferentes combinaciones,

mientras uno de 3X3, el más popular, tiene aproximadamente 6.7 x 10^{21} posibles configuraciones; y el de 4x4, tiene aproximadamente 6 x 10^{98}. Claramente encontrar la solución usando el método de fuerza bruta es imposible. Este es otro nicho donde se requieren técnicas de IA para la solución.

De manera que el **nicho de aplicación de la IA** está en la predicción del comportamiento de los sistemas complejos y en la solución de sistemas cuya complejidad computacional sea muy alta.

5. Herramientas de IA

Considerando los enfoques para afrontar la IA, se han desarrollado muchas herramientas que podemos clasificar de la siguiente forma:

Orientadas al comportamiento. Entre otras muchas las más destacadas son:

- Autómatas celulares
- Algoritmos genéticos
- Programación genética
- Lógica difusa

Con base en Redes neuronales Artificiales:

- Redes neuronales artificiales
- Machine Learning
- Deep Learning
- IA Generativa

5. Ejercicios

1.1 Pídale a varios LLMs de IAGen una definición de IA, compárelas con lo visto en el capítulo.
1.2 Dé una definición propia de inteligencia artificial.
1.3 Vea el video que se encuentra en https://www.youtube.com/watch?v=JsmKUCiPHUY&t=7s . Haga un análisis de lo discutido en el documento con respecto al video.
1.4 Haga un resumen sobre Cosmos de NVIDIA
1.5 Investigue sobre los planes del gobierno con respecto a la IA.

8. Bibliografía.

Adami, C., Introduction to Artificial Life, Springer-Verlag, 1998.
Brainterberg V, Vehicles, Experiments in Synthetic Psychology, MIT Press
Franco O., Escallon S., Estudio Básico de Vida Artificial, Facultad de Ingeniería, Universidad Nacional, Bogotá, 1995.
Both A., Kuhn W., Matt Duckham, Spatiotemporal Braitenberg Vehicles.
Gribbin J., Deep Simplicity, Random House, 2004.
Langton, Cristopher, Artificial Life, an Overview, A Bradford Book, MIT Press, 1995.
Legg S., Hutter M., A Collection of Definitions of Intelligence, IDSIA Switzerland, 2007.
Martínez J. J., Vida artificial, nuevos conceptos para la ingeniería, Revista Facultad de Ingeniería, Universidad Nacional, Bogotá, 1999.
Martínez J. J., *Vida Artificial*. Revista de la Facultad de Medicina, Universidad Nacional, Bogotá, Volumen 46, No. 2. Abril - Junio 1998.
Mijwel M. M., History of Artificial Intelligence, Universidad de Bagdad, 2015.

Mordechai Ben Ari, F. Mondada, Elements of Robotics, Springer Open, 2018
https://www.youtube.com/watch?v=-lnHHWRCDGk

---

[i] https://www.historyofdatascience.com/dartmouth-summer-research-project-the-birth-of-artificial-intelligence/

Con respecto al desarrollo del curso:

1. Crear un grupo de whatsApp.
2. Cuando se utilicen herramientas de IA Generativa, se deben hacer comentarios sobre que herramientas utilizaron, los prompts y hasta qué punto les sirvió para el desarrollo de la documentación, los ejercicios o proyectos.
3. Crear un sitio en GitHub para la entrega de los documentos y los programas.
4. Los programas deben presentarse y documentarse en Jupyter Notebook.

2. Autómatas Celulares

José J. Martínez P.
jmcursosingenieria111@gmail.com
Febrero 2026

2.1 Introducción.

El inicio de los autómatas, ACs, se da en los 40s**,** cuando John von Neumann**,** uno de los padres de la computación, buscaba la auto-reproducción de robots, junto con el matemático Stanisław Ulam**,** en el Laboratorio Nacional de Los Álamos, diseñó el primer AC teórico como una rejilla bidimensional**,** donde cada celda cambia de estado aplicando reglas simples considerando su estado y el de sus celdas vecinas.

Los autómatas celulares no generaron atención sino hasta los 70s, cuando el matemático británico John Conway desarrolló Life, el Juego de la Vida. Con este juego demostró que con reglas muy simples se podían generar comportamientos sorprendentemente complejos y caóticos, por lo que se volvió muy popular, en la academia.

En los 80s, Stephen Wolfram**,** físico y matemático, impulsó una revolución en el estudio de los AC unidimensionales. Clasificó sus comportamientos y propuso que podían ser una base alternativa para describir la naturaleza, desarrollando esta idea en su libro "A New Kind of Science" en 2002.

Como vimos, los sistemas complejos, son sistemas o fenómenos cuyo comportamiento cambia en el tiempo, son dinámicos. En estos sistemas, su estado en un momento dado se puede calcular a partir del estado anterior y de sus interacciones con el ambiente. Estos sistemas, que generalmente son continuos, los podemos tratar de forma discreta, tanto temporal como espacialmente.

Vamos entonces a hablar de los Autómatas Celulares, ACs, sistemas complejos discretos. Decimos entonces que los ACs son sistemas dinámicos discretos con un número enumerable de estados. En AC la idea básica es describir cualquier sistema complejo, a partir del comportamiento de sus componentes elementales (en teoría homogéneas), que llamamos células, que crean la complejidad a partir de la aplicación de reglas de interacción elementales entre las células. Los ACs se preocupan por la pregunta de cómo surge el comportamiento emergente en un sistema partir de su actualización, aplicando reglas que relacionan una célula con sus vecinos cercanos.

En otras palabras, se deja que la complejidad del sistema **emerja** a partir de la interacción de las células, que siguen, en principio, reglas simples. En inglés, un autómata es *automato*n y un grupo de autómatas es *automata*. De manera que el término en inglés *Cellular Automata*, da más claridad, sobre el hecho de que la complejidad emerge a partir de un grupo de autómatas elementales, pues en español usamos el mismo término para los autómatas elementales como para el grupo de autómatas.

2.2 Vecindad

"Dime con quién andas y te diré quién eres"
Refrán popular

El concepto fundamental en los AC es el concepto de vecindad. Se trata de que en el tiempo, el comportamiento o estado de una célula está determinado por el comportamiento de la misma célula y el de sus células vecinas, por lo que per se son sistemas complejos. Es por esto importante hacer algunas definiciones:

Vecindad, es el conjunto de vecinos de una célula que influye en su comportamiento. En su definición común, es el conjunto de los vecinos de una población, barrio, calle o casa. Ej. A la fiesta decembrina asiste toda la vecindad. Pero además, en general, una determinada vecindad tiene costumbres comunes, sigue reglas comunes con algunas excepciones.

Como se puede observar, son muchos los comportamientos que se generan a partir de una vecindad, tantos que en general definen el comportamiento de muchas personas. Es así que, en nuestra casa, nuestro trabajo, nuestras clases, nuestro desplazamiento, nuestras reuniones, siempre hay una serie de reglas implícitas, de las cuales generalmente no somos conscientes, que si bien no están formalmente establecidas, definen nuestro comportamiento. Otros ya los definen claramente las leyes y las normas aprobadas. En otras palabras, nosotros, las personas, también somos autómatas celulares. ¿Alguna relación con Matrix?

2.3 Definición Formal

Un autómata celular D-dimensional (o AC-D), es una secuencia C_t, definida por una 5-tupla, $(L, w, U, f,\ C_0)$ donde:

L: es un retículo,
w: es un alfabeto, que representa los estados de la celda.
$U = (u_1,\ u_2, \dots, u_n)$ es una secuencia finita de elementos del retículo
$f: w^U \rightarrow\ w$, es la función de transición o regla local
$C_0 : L\ \rightarrow w$, es alguna configuración inicial.

Ejemplo 1. AC 1D, autómatas celulares de una dimensión.

Retículo: L

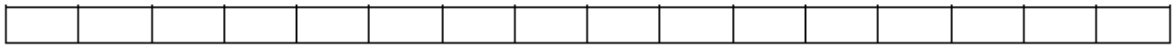


Figura 2.1, retículo o universo para los autómatas celulares de una dimensión

Alfabeto: $w = \{0, 1\}$

Vecindad: U

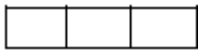


Figura 2.2. Vecindad, radio r=1 para autómatas AC-1D. Se Pueden definir otras vecindades

Configuración inicial del autómata, aleatoria: $C_0 : L\ \rightarrow w$

1	1	1	1	0	1	1	0	0	1	0	1	0	0	1	0

Figura 2.3. Estado inicial de un AC-1D, r=1, generalmente es aleatorio

En la gran mayoría de casos, el AC-1D funciona como un collar, es decir el autómata del extremo izquierdo se une con el autómata del extremo derecho. Cuando se tiene otro tipo de retículo esta configuración se denomina toroidal.

Transición local o regla de comportamiento local:

$$c_c(t+1) = f(c_{izq}(t), c_{cc}(t), c_{der}(t))$$

Donde c_c es la celda central, c_{izq} es la celda de la izquierda y c_{der} es la celda de la derecha.

Nota. La $\boldsymbol{C}$ mayúscula hace referencia a todo el Autómata Celular, el retículo; la c hace referencia a uno de los autómatas, el cuadrado pequeño.

Ejemplo 2. AC-2D, autómatas celulares de 2D:

Retículo: L

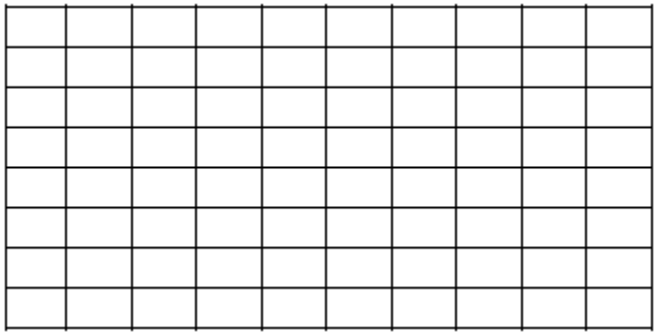


Figura 2.4. Retículo o universo para los AC 2D. Pueden conformarse otros tipos de retículos.

Alfabeto: $w = \{0, 1\}$

Vecindad: U**,** en AC-2D las vecindades comunes son

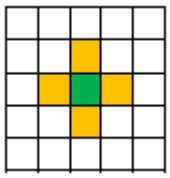


Figura 2.5, vecindad de von Newmann, tiene 5 vecinos

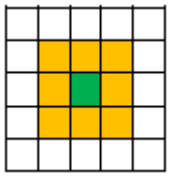


Figura 2.6, vecindad de Moore 9 vecinos, es la más utilizada.

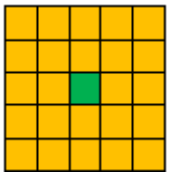


Figura 2.7, vecindad de Moore extendida 25 vecinos.

Configuración inicial del autómata, aleatoria: $C_0 : L \rightarrow w$

1	0	1	1	0	0	1	1	1	1
1	1	0	1	0	1	1	1	0	1
1	1	0	1	1	1	0	1	0	0
0	0	1	0	1	0	1	0	1	0
1	1	0	1	0	1	0	1	1	0
0	0	1	0	1	0	1	0	1	0
1	0	1	1	1	0	0	1	0	0
0	1	0	0	0	1	1	0	1	1

Figura 2.8, configuración aleatoria inicial de un AC 2D.

Transición local o regla de comportamiento local, para un AC 2D con vecindad de Moore:

$$c_c(t+1) = f(c_N(t), c_{NE}(t), c_E(t), c_{cc}(t), c_{SE}(t), c_S(t), c_{SO}(t), c_O(t), c_{NO}(t))$$

Donde c_c es la celda central, c_N es la celda norte, c_{NE} es la celda noreste, c_E es la celda este, c_{SE} es la celda sureste, c_S es la celda sur, c_{SO} es la celda suroeste, c_O es la celda oeste y c_{NO} es la celda noroeste.

2.4 Propiedades Esenciales de un Autómata Celular

1. Universo: Un *retículo* o arreglo *regular*, también se le conoce como teselación, donde cada *célula* tiene un estado discreto.
2. Dinámico: Un comportamiento dinámico que describe la regla local.
3. Homogeneidad: Todas las células se actualizan con la misma regla local
4. Paralelismo: El estado de todas las células se actualiza simultáneamente. La regla define el estado de cada célula central en el siguiente paso de tiempo, dependiendo de los estados de las células vecinas y de su propio estado.
5. Localidad: las reglas son locales por naturaleza.

Para construir un Autómata Celular, se deben definir: la célula; el retículo; la regla; y la vecindad

1. La Célula.

- Es el elemento básico de un AC, se comporta como un autómata de estado finito, sus entradas son el estado de las células vecinas y el suyo propio; su salida, su nuevo estado.
- Una **célula** es una clase de elemento de memoria que almacena **estados**. El número de estados o conjunto de estados se representa por ω.

- En el caso más simple, ω = {0, 1}. En casos más complejos las células pueden tener más estados diferentes.

2. El retículo.

- Es el espacio donde se conforma el AC. El retículo más simple es el de *una dimensión* con una configuración *cuadrada, 1D.* Esto significa que el AC es como un collar de perlas.
- Los AC más comunes son de 1D o 2D, aunque en ciertas aplicaciones se trabajan AC 3D. También se conoce como teselación.

3. *Las reglas*

- Para introducir la *dinámica* al sistema, se le deben definir las **reglas** al AC.
- En algunas aplicaciones son ecuaciones en diferencia.

3. *La vecindad.*

- La vecindad se establece dentro del retículo, casi siempre con vecindades regulares.
- La vecindad, en vecindades regulares, se conforma con base en el radio ***r***, que define el horizonte del autómata.
- En ciertas aplicaciones puede uno definir su propia vecindad. Por ejemplo, en sistemas de criptografía se toman vecindades completamente irregulares, o en los diagramas de Voronoi

Un ejemplo de dinámica "microscópica" que resulta de la interacción local es la "ola" en un estadio de fútbol. Cada persona reacciona dependiendo del "estado" de sus vecinos. Si se ponen de pie y alzan los brazos, usted también lo hace, después de un corto tiempo; sus vecinos se vuelven a sentar, y usted se sienta siguiendo lo que hacen sus vecinos.

2.5. Autómatas Celulares de 1D

2.5.1 Autómatas Celulares Simples

Vamos a estudiar ACs 1D con r=1, es decir una vecindad de tres células, w = {0, 1}. En estos autómatas, hay 2^8 = 256 reglas que han sido estudiadas con mucho detalle por Stephen Wolfram.

Figura 2.9, Las 256 reglas de un AC 2D.

Aunque algunas reglas generan comportamientos globales similares, en general el comportamiento del AC depende de sus condiciones iniciales y de su regla de comportamiento local. Cada regla se identifica con base en un número entre 0 y 255.

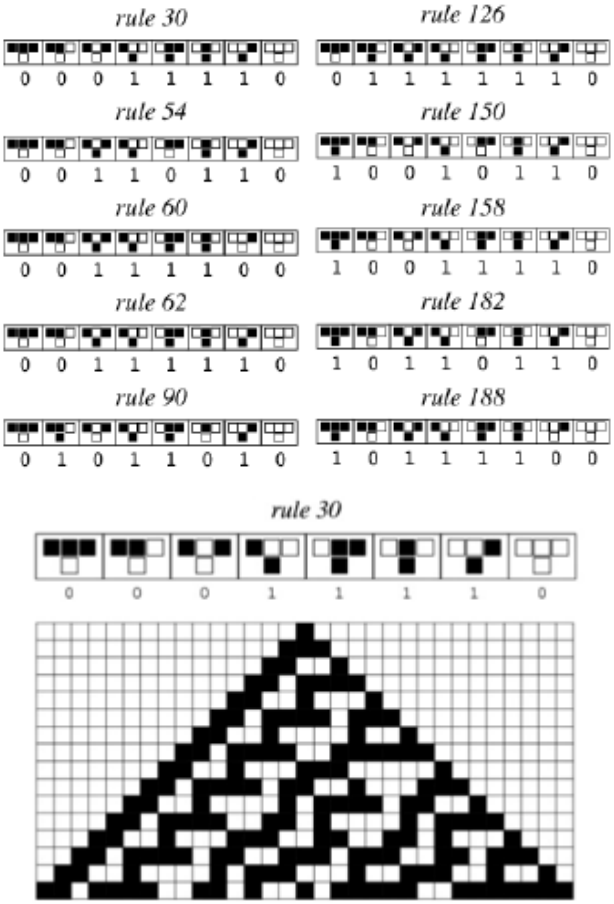


Figura 2.10, Algunas reglas de un AC 1D.

De la figura 2.10, el número de la regla se encuentra tomando la cadena de ceros y unos de la regla, así la regla 30 tiene una cadena 00011110, que se toma como un polinomio en base 2 que al evaluarse nos da el 30.

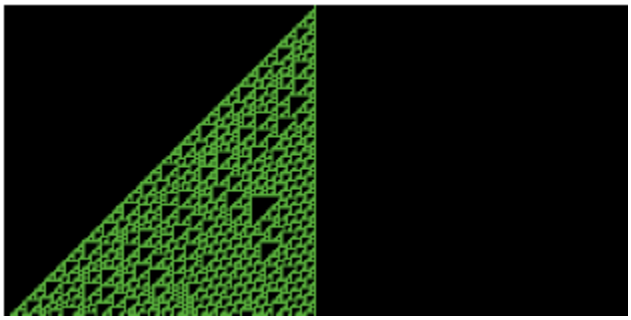


Figura 2.11, Regla 110, tiene un comportamiento complejo[1].

Por su comportamiento, también son de interés las reglas: 23, 25, 27, 30, 31 y otras, ver[2].

2.5.2 Reglas Totalísticas

En este caso el estado siguiente de la célula central depende únicamente de la suma de los estados de las células vecinas, también se conocen como reglas de voto. Formalmente, la regla global de un AC totalístico 1D está definida por:

$$c_c(t+1) = f(c_{izq}(t) + c_{cc}(t), + c_{der}(t))$$

2.5.3 Comportamientos de Wolfram

En el estudio formal desarrollado por Wolfram de los AC 1D, se encontró que los comportamientos de los AC 1D, se podían clasificar en una de cuatro categorías diferentes:

Clase I. Los ACs de la Clase I siempre se resuelven en un patrón homogéneo global, donde cada célula termina quedándose en el mismo estado indefinidamente. Se pueden considerar que tienen un orden perfecto. Ejemplos son las reglas 0, 32, 160, 249.

Clase II. Los ACs de la Clase II forman estructuras que convergen rápidamente a estados periódicos que se repiten a través de un número fijo de estados. Hay orden, sin embargo, recorren rutas periódicas. Los ejemplos más simples de esta clase son los que alcanzan algún estado invariable global, como la regla 4 o la regla 12. Otros ejemplos son las reglas 4, 108 y 218. Aunque también se considera de esta clase la regla 24.

Clase III. Los ACs de la Clase III se caracterizan por su naturaleza caótica pues producen diseños aleatorios, no hay un orden. Como el ruido en una pantalla de rayos catódicos sin señal. El ejemplo clásico es la regla 30, otros ejemplos son las reglas 22, 30, 90, 126. Y 150. Aquí se puede apreciar que un pequeño cambio en las condiciones iniciales, eventualmente afecta todas las partes del sistema.

Clase IV. Los ACs en la Clase IV se encuentran en el límite el orden de la clase II y el caos de la clase III. Forman estructuras complejas, pero con estructuras localizadas que "migran" dentro del diseño principal. Dependiendo de la configuración inicial, los diseños de la Clase IV se resuelven eventualmente en diseños de Clase I o Clase II. Ejemplos son las reglas 54 y 110.

Las clases I y III no son interesantes, mientras que las clases II y IV tienen propiedades muy interesantes desde el punto de vista de la complejidad y de la vida. Se supone que la vida requiere orden para almacenar experiencia y caos para interactuar con el ambiente. Es por esto por lo que se dice que la vida está al borde del caos, debe tener un orden mínimo para almacenar experiencia y manejar el caos para adquirir esa experiencia.

---

[1] Simulada en NetLOgo

[2] http://mathworld.wolfr am.com/ElementaryCellularAutomaton.html

Para un AC con k número de estados de la célula y n número de vecinos incluida la célula central, el número de posibles reglas se puede calcular como k^{k^n}. Si suponemos un autómata con 4 estados y nueve vecinos, el número posible de reglas seria 4^{4^9}, o 4^{262144}. Un número gigantesco. De ahí la complejidad del comportamiento de los autómatas celulares.

2.6 Autómatas Celulares 2D

Los AC 2D con un retículo cuadrado, permiten el manejo de un gran nivel de complejidad y por eso hay mucho interés en sus estudio y aplicación. Ya habíamos mencionado las vecindades y dentro de ellas la más utilizada es la vecindad de Moore.

2.6.1 LIFE de Conway

Uno de la AC de 2D más estudiado y conocido es “LIFE”, desarrollado por John Conway. Se considera un buen ejemplo de emergencia y auto-organización, por la cantidad de objetos que se van produciendo en los ciclos de ejecución, dependiendo de su configuración inicial. Se ha probado que es una máquina de Turing, por lo cual es un computador universal.

LIFE comenzó a finales de los 1960s, principios de los setenta cuando Conway estableció una simplificación de las investigaciones sobre autómatas celulares de John von Neumann. Conway había querido encontrar el AC más simple que soportara la computación universal. Para esto decidió usar dos estados, muerto o vivo, y cuatro reglas simples:

- Si una célula viva tiene menos de dos vecinos vivos, se muere. Soledad.
- Si una célula viva tiene dos o tres vecinos, sigue viva. Felicidad.
- Si una célula viva tiene más de tres vecinos, se muere. Sobrepoblación.
- Si una célula muerta tiene exactamente tres vecinos, nace. Reproducción.

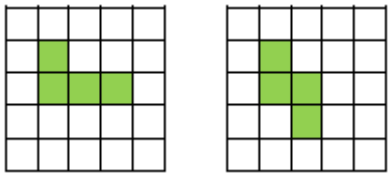


Figura 2.12. Si una célula viva tiene menos de 2 vecinos, se muere.

Como con los ACs 1D, estas reglas simples generan un comportamiento increíblemente complejo. A continuación, se presenta un "deslizador" (glider), una estructura dentro de LIFE que tiene un ciclo de 5 estados, y un movimiento abajo a la izquierda en cada ciclo.

Figura 2.13. Como se desplaza un deslizador.

Life ha sido trabajado mucho en física teórica y ha habido muchos congresos internacionales, pues se comporta como un computador universal, según lo planteado por Turing.

2.7 Autómatas celulares probabilísticos

Son autómatas que cuyas reglas permiten el cambio de estado de un autómata con base en una probabilidad, permitiendo movimientos aleatorias. Este tipo de autómatas se utiliza en la difusión de un virus en un ambiente, como el caso del virus HIV o del COVID-19, o de incendios forestales.

Para ver cómo trabaja un AC probabilístico, veamos la como se aplica en la construcción de un modelo con reglas probabilísticas para estudiar el comportamiento del CPVID, utilizando números aleatorios con base en una distribución uniforme:
Suponga que tiene un AC 2D, en un retículo cuadrado de 10.000 Celdas; y que w = {Np, Ps, Cs, Cns, Ct, Pac}, donde:

- Np=No hay nada en la celda;
- Ps=Personas sanas;
- Cs=Persona con Covid-19 sintomática;
- Cns=Persona con Covid-19 asintomática;´
- Ct=Persona con Covid-19 en cuarentena o en cuidados médicos;
- Pac=Persona curada con anticuerpos.
- M=Persona muerta, no aparece en el retículo

Inicialmente el retículo se carga con 5000 Ps distribuidos uniformemente, con 5 Cs y 5 Cns también distribuidos aleatoriamente. Se usa la´vecindad de Moore.

Reglas.

- Si Ps tiene n vecinos pasa con probabilidad 0.n a ser Cns con probabilidad 0.4; o Cs con probabilidad 0.7.
- Cs después de 7 ciclos pasa a Ct
- Cns después de 30 ciclos pasa a Pa.
- Ct después de 20 ciclos con probabilidad 0.8 pasa a Pac o con probabilidad 0.2 pasa a M, que no se representa en el retículo.

Una vez se han ejecutado estas reglas en paralelo, se desplaza cada celda a una celda Norte, Sur, Este u oeste con probabilidad 0.25, si la celda está vacía. En caso contrario, se debe llevar el número de ciclos para las celdas Cs, Cns y Ct, lo mismo que el conteo de M. En cada ciclo se debe llevar el conteo de cada tipo de celda. Con la simulación se busca responder las preguntas siguientes: ¿Después de cuantos ciclos se estabiliza la población? ¿Cuantas celdas de cada estado quedan? ¿Cuántos muertos hubo?

2.8 Aplicaciones de los Autómatas Celulares

- **Epidemiología**

Así como vimos la simulación del COVID, podemos aplicar los AC a diferentes enfermedades contagiosas.

- **Antropología**

Se utilizan los ACs para hacer representaciones espacio-temporales para modelar la formación de sociedades civiles.

- **Sociología**

Consideramos que la simulación más cercana a la realidad de los sistemas sociales se logra con ACs, como por ejemplo para el estudio de las causas y efectos de la violencia civil. Se supone que muy pronto en la formación de profesionales en este tipo de áreas de las ciencias sociales, debe incluir estos tipos de simulación.

- **Criptografía**

En el caso de los criptosistemas, se obtienen sistemas seguros frente a ataques de fuerza bruta y texto elegido. En este caso se utilizan AC **reversibles** que después de un determinado número de pasos vuelven a su configuración inicial. Es decir, el tiempo de evolución del autómata es completamente reversible. Eso significa que, en cualquier momento, en el desarrollo de las reglas, se puede ir hacia adelante o hacia atrás en el tiempo *sin perder información*.

Es un campo de desarrollo muy importante dado que la transformación del mensaje no es una transformación lineal, un cambio en un bit del mensaje genera una cascada de cambios en el mensaje codificado. Ejemplo CAFÉ, Un criptosistema de clave simétrica basado´en A.C.s.

- **Biología**

Se pueden simular varios procesos y sistemas con AC, desde el crecimiento de tejidos celulares, patrones de las pieles de varios animales, modelos presa-predador y comportamientos de neuronas cerebrales.

Figura 2.14, a la izquierda un tigre con su piel, como resultado de un autómata celular; a la derecha un autómata celular 3D.

Muchos modelos dinámicos interesantes provienen de la biología, cuando interactúan dos especies predador y presa. Este es un ejemplo de modelo predador-presa muy interesante. El universo definido en este modelo se compone de tiburones, peces y vacío. Por otra parte, se introducen funciones aleatorias, es decir si además se cumplen ciertas condiciones, se sigue una regla con cierta probabilidad. Es por lo menos inquietante la piel de algunos animales que casi corresponden a autómatas celulares, en la gráfica 2.14, se puede uno preguntar ¿son ambos animales?

Reglas. Son varias las reglas que describen el comportamiento de cada célula:

- Nacimiento, longevidad y muerte de un pez o un tiburón
- Reproducción de peces y tiburones

- Sobrepoblación o subpoblación
- Interacción pez-tiburón

- **Flujo vehicular**

Se emula el desempeño de un tramo vial, considerado como un A.C., considerando el espacio entre vehículos, el desempeño de un conductor y la dinámica de su vehículo y de los vehículos que comparten el tramo,

- **Simulación de fenómenos naturales**

El modelado de sistemas físicos complejos a través de simulación por computador es una herramienta común para comprender el mundo natural. La complejidad de estos modelos a veces oscurece los principios físicos básicos subyacentes en el fenómeno modelado.

En este contexto se busca simular fenómenos naturales en donde su comportamiento se rige por ecuaciones diferenciales ED o por la interacción local de sus componentes. En el primer caso las ecuaciones ED se convierten en ecuaciones en diferencia EDF, para la discretización del sistema, y se aplican a cada autómata; permitiendo además, agregar algunas características que no permiten que haya inestabilidad estructural. De este modo se han podido modelar campos electromagnéticos, mecánica de fluidos, distribución de calor en placas, entre otros.

Veamos el caso de la transmisión de calor en una barra. Supongamos que se tiene la ecuación diferencial genérica:

$$\frac{dx}{dy} = f(x, y)$$

Lo primero es transformar la ED en EDF, como:

$$x(t+1) = x(t) + f\big(x(t), Y(t)\big)$$

Luego, se reemplazan todas las variables reales x y y por el estado discreto de las variables del AC, entonces el límite de estas variables tiene un alcance finito. De manera que las variables de estado se muestran como conjuntos pequeños finitos.

Veamos un ejemplo con la ecuación de difusión de calor, en 1D:

$$\frac{dT}{dt} = k\frac{d^2T}{dx^2}$$

Donde T es la temperatura y k es la difusividad térmica. La temperatura es una función con valores reales y es continua para cualquier valor de cualesquiera sean las condiciones iniciales. La discretización de la ecuación anterior es:

$$T_i^{n+1} - T_i^n = \frac{k\Delta t}{\Delta x^2}\,(T_{i+1}^n - 2T_i^n + T_{i-1}^n)$$

Donde i y n son los índices espaciales y temporales. Si se escogen los pasos de espacio y de tiempo tal que $\frac{k\Delta t}{\Delta x^2} = \propto$ y haciendo transformaciones nos lleva a la ecuación:

$$T_i^{t+1} = \propto\,\left(T_{i+1}^t - 2T_i^t + T_{i-1}^t\right) + T_i^n$$

Ahora, suponga que tiene una barra de longitud L cuyo extremo izquierdo se pega a una pared que está a 150°C. Suponga que la temperatura promedio del ambiente es de 15°C. La barra tiene L = 1m, α = 0.5, divídala en 4 segmentos y diga cuál es la temperatura en el segmento final después de 4 pasos de tiempo.

- **En robótica**

Aunque esta herramienta no se ha utilizado mucho en robótica y en los campos de la producción industrial, o por lo menos se desconoce su uso, vamos a ver un ejemplo donde se puede ver su gran potencialidad.

Se tiene un robot móvil simple equipado con sensores de proximidad y dos motores que pueden girar en ambos sentidos. En este caso el uso de los ACs está dirigido a que el robot evite los obstáculos que se le puedan ir presentando en el camino, eliminando las colisiones.

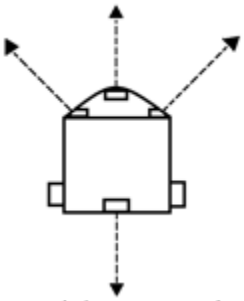


Figura 2.14. Modelo esquemático del robot.

El robot se modela como dos autómatas celulares no uniformes, con una vecindad de Von Neumann. Es un AC no uniforme porque los cambios en los estados de las cuatro células externas, dependen de la información que proveen los sensores. Mientras que la célula central, que representan los motores izquierdo y derecho, cambia de estado dependiendo de los estados de las células anteriores, siguiendo una regla.

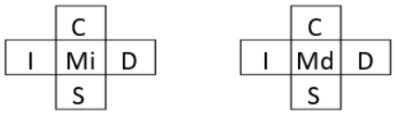


Figura 2.15. Vecindad de los autómatas.

Donde C, D, I, S, son células pasivas que simbolizan los sensores centro, izquierdo, derecho, y sur, las células Mi, Md, son células operacionales activas que representan el motor izquierdo y el motor derecho respectivamente.

Veamos un ejemplo en el que solo se trabajan las células C, D, I; que corresponden a los sensores de distancia del robot. Haciendo $k = C, D, I$; los estados $S^{k} = \{0,1,2\}$ de estas células que tienen un significado particular: el estado cero es un intervalo entre 0 y 8 cm; el estado 1, un intervalo entre 8 y 16 cm; y el estado 2, un intervalo mayor a 16 cm.

Con respecto al estado de las células de los motores, también tienen un significado particular, así los estados de los motores izquierdo y derecho $S^{Mi} = S^{Md} = \{0,1,2\}$, el estado 0, representa el motor apagado; el estado 1, motor encendido, con giro hacia adelante; y el estado 2, motor encendido con giro hacia atrás.

Tomando η_x^t para simbolizar los estados de la vecindad de la célula activa x en el tiempo t, obedece a la regla de transición $\Phi(\eta_x^t)$, que lo lleva al estado $S_x^{t+1} = \Phi(\eta_x^t)$. Considerando que se tienen tres estados para los tres sensores considerados, se tiene un total de 27 reglas.

La tabla de las reglas para los dos autómatas es la siguiente:

M_I	C	D	I	S_{MI}^{t+1}		M_D	C	D	I	S_{MD}^{t+1}
*	0	0	0	2		*	0	0	0	2
*	0	1	0	0		*	0	1	0	2
*	0	2	0	2		*	0	2	0	0
*	0	0	1	2		*	0	0	1	0
*	0	1	1	2		*	0	1	1	0
*	0	2	1	2		*	0	2	1	0
*	0	0	2	2		*	0	0	2	0
*	0	1	2	2		*	0	1	2	0
*	0	2	2	2		*	0	2	2	0
*	1	0	0	2		*	1	0	0	0
*	1	1	0	0		*	1	1	0	2
*	1	2	0	0		*	1	2	0	2
*	1	0	1	2		*	1	0	1	0
*	1	1	1	1		*	1	1	1	0
*	1	2	1	1		*	1	2	1	0
*	1	0	2	2		*	1	0	2	0
*	1	1	2	0		*	1	1	2	2
*	1	2	2	0		*	1	2	2	2
*	2	0	0	0		*	2	0	0	2
*	2	1	0	1		*	2	1	0	0
*	2	2	0	1		*	2	2	0	0
*	2	0	1	0		*	2	0	1	1
*	2	1	1	1		*	2	1	1	0
*	2	2	1	1		*	2	2	1	1
*	2	0	2	0		*	2	0	2	1
*	2	1	2	1		*	2	1	2	1
*	2	2	2	1		*	2	2	2	1

Tabla 2.1. Reglas de transición de los autómatas. * significa que no importa el estado.

Sin embargo, se pueden obtener reglas con un comportamiento mucho mejor, usando para encontrarlas Algoritmos Genéticos. Con este ejemplo, aunque simple, se permite entrever las muchas aplicaciones que se pueden desarrollar en el campo de la robótica con los ACs.

- **Estudios Teóricos**

En este campo se utilizan los AC para el estudio de temas como complejidad, computación universal, computación en paralelo, síntesis de sistemas, auto-organización, emergencia, entropía, teoría de

lenguajes computacionales o estudio de fractales, biología computacional, ciencias sociales entre otros.

- **Otras aplicaciones**

Entre otras aplicaciones que se han trabajado están: la simulación de crecimiento del cáncer, los modelos de uso de la tierra, los modelos de incendios forestales, modelos de reacciones químicas, los patrones de pigmentación de piel, crecimiento de conchas marinas y corales, comportamiento de colonias de microorganismos, modelo de simulación de evacuaciones, entre otros.

El problema inverso

Como problema inverso se plantea el caso en que se conoce un comportamiento a nivel macro, a nivel del SCA, y se desea encontrar las reglas que generan ese comportamiento. Este problema tiene muchas implicaciones importantes dado que en general estamos observando dinámicas de comportamiento que no comprendemos y que queremos explicarnos, no es un problema simple de resolver y en muchos casos se considera un problema abierto.

En "Prototipo de un Sistema Clasificador para Búsqueda de Reglas de Autómatas Celulares en 2D[3]", se presenta un ejemplo de búsqueda de las reglas de LIFE de Conway a través de un Sistema Clasificador de Holland.

2.9 Diagramas de Voronoi

¿Pero qué pasa con la vecindad si la telesación no es regular? Vemos los planos de una ciudad o de un departamento, o de una región y no encontramos una forma regular de universo como las que se han definido, para poder utilizar su potencialidad en este tipo de escenarios reales. Una forma de hacerlo es utilizar los diagramas de Voronoi.

Los **diagramas de Voronoi** tienen una amplia variedad de aplicaciones en múltiples campos, veamos algunas:

En gráficas para generar texturas naturales, en simulación de fracturas; en búsqueda y optimización como planear rutas en zonas de influencia, en agricultura de precisión, en procesamiento de imágenes, en diseño de redes inalámbricas. En resumen, los **diagramas de Voronoi** se pueden aplicar cuando hay que dividir un espacio en zonas de influencia según la proximidad a un conjunto de puntos.

[3] Prototipo de un Sistema Clasificador para Búsqueda de Reglas de Autómatas Celulares en 2D, L. R. Benavides y O. I. Triana, Facultad de Ingeniería U.N., Bogotá, 2002.

Figura 2.16. Teselación irregular con diagramas de Voronoi.

2.9 Autómatas Celulares 3D

Suponiendo un universo como un cubo, podemos definir nuestros AC 3D. Nos encontramos con que casi todas las reglas de transición que se usan en AC 2D se pueden usar en AC 3D, la gran diferencia está en que la vecindad de los AC 3D es mucho más grande.

Si se trabajan los AC 1D en 3D, con r=1, y = {0, 1}, cada autómata es un cuadro en un plano y la otra dimensión la da el tiempo. Se tienen ahora 4294967296 autómatas y 3283936144 reglas. En la figura 5.14, se presenta una extensión de la regla 30 AC 1D a 3D.

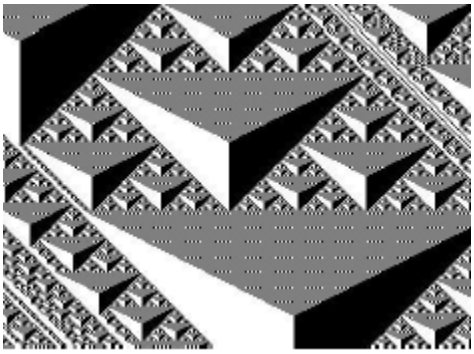


Figura 2.17. Extensión de la regla 30 AC 1D a 3D[4].

Con el desarrollo de las impresoras 3D, se pueden generar patrones emergentes estocásticos. Las impresoras 3D se pueden considerar como autómatas celulares asincrónicos 3D, pues generan patrones 3D acumulando patrones 1D, el caso más simple de un AC 1D asincrónico. Esto nos permite ampliar nuestras posibilidades de utilizar los AC 3D en muchas otras aplicaciones.

[4] https://nbickford.wordpress.com/2010/05/19/cellular-automata/

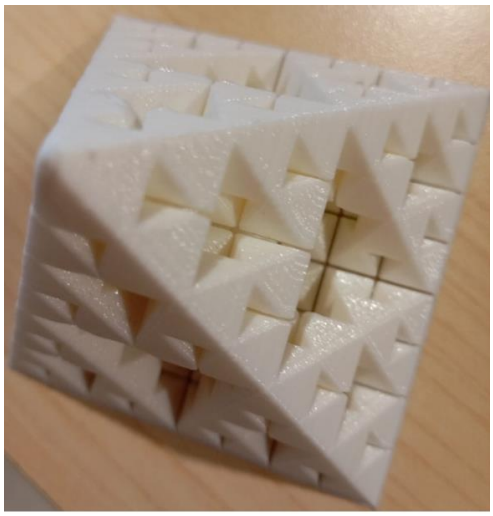


Figura 2.18. Triangulo de Sierpinsky impreso en 3D.

2.10 Ejercicios y Problemas.

1. Observe sus comportamientos en la casa, en la universidad y en el medio de transporte que utiliza. Encuentre, para cada uno de estos escenarios sus reglas básicas.
2. Suponga una enfermedad, o un incendio forestal, o una moda, desarrolle un modelo de difusión usando ACs probabilísticos. O simule un robot con dos ruedas que evite obstáculos.
3. Simule el comportamiento de un robot con tres sensores de distancia, que recorre un espacio bidimensional, donde hay 4 objetos distribuidos aleatoriamente, que no se choca con esos objetos.
3. Tome el plano de una ciudad pequeña y localice, por ejemplo, las droguerías, centros de atención de salud y colegios. Por cada concepto dibuje un diagrama de Voronoi. ¿Considera que puede faltar una droguería, o un centro de atención de salud o un colegio? ¿Hay alguna relación entre los diagramas?

2.10 Bibliografía

Cortes, C., P., y Cubillos, M. G., Desarrollo de Software Adaptativo para el Aprendizaje de Robots Móviles, Basado en Autómatas Celulares, Facultad de Ingeniería, Universidad Nacional, Bogotá, 1999.

Kanada Y, 3D Printing and Simulation of Naturally-Randomized Cellular-Automata, AROB, 2014

Mitchell Melanie, Introduction of Complexity, MOOC,2015

Martínez José J., Notas Vida Artificial, 2009.

Ortiz Z, Martínez J., Cifuentes J., Cortés E, Ortíz E, CAFE: Un Criptosistema de Clave Simétrica Basado en Autómatas Celulares, Congreso Colombiano de Inteligencia Computacional, 2009.

Weisbuch G, Rickebush S, Complex System Dynamics, Santa Fe Institute, 1990.

Wolfram, S. *A New Kind of Science.* Champaign, IL: Wolfram Media, 2002.

http://mathworld.wolfram.com/ElementaryCellularAutomaton.html

http://natureofcode.com/book/chapter-7-cellular-automata/

https://argentinainvestiga.edu.ar/noticia.php?titulo=un_modelo_de_simulacin_de_evacuaciones_que_emplea_autmatas_celulares&id=2142 Sobre modelos de evacuación.

https://www.youtube.com/watch?v=qjEXwJScXyw Sobre diagramsa de Voronoi.

3. Algoritmos Genéticos

José J. Martínez P.
Jmcursosingenieria111@gmail.com
Febrero 2026

Entre los diferentes enfoques para atacar el problema de la IA, se presentó el enfoque de vida, los sistemas bioinspirados, en este caso del proceso evolutivo. Por otra parte, hablábamos de que el nicho de la IA era solucionar problemas con alto grado de complejidad computacional. El Algoritmo Genético es una de una serie de herramientas bioinspiradas entre las que se encuentran: Optimización por Colonia de Hormigas, Recocido Simulado, Inteligencia de Enjambre, Búsqueda Tabú, Lobo Gris (Gwolf) y otras, que se han desarrollado para la solución de *problemas de optimización y* de *alta complejidad computacional*. En la figura 3.1 se presenta una comparación de popularidad de estos algoritmos.

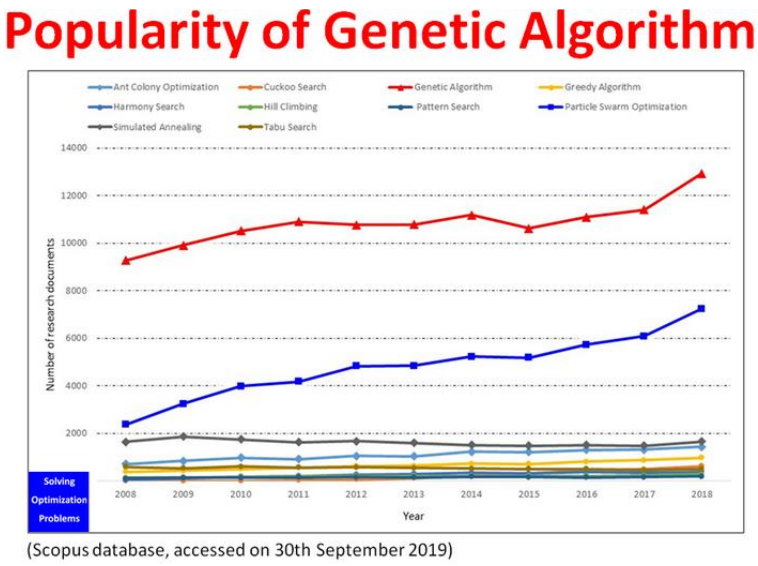


Figura 3.1. Popularidad de los Algoritmos Genéticos frente a otros algoritmos de optimización

De la figura 3.1, se puede observar que el algoritmo genético es la herramienta más trabajada para la solución de problemas especialmente con alta complejidad computacional y de optimización. Esta es una de las razones para trabajar esta herramienta en el curso.

3.1 **Introducción a los Algoritmos Genéticos**

En los 1950s y 1960s varios investigadores estudiaron independientemente la evolución natural con la idea de que sus mecanismos se pudieran usar como una herramienta de optimización para

solucionar problemas de ingeniería. La idea en todos estos sistemas era evolucionar una población de soluciones candidatas usando operadores inspirados en la variación genética y la selección natural, para solucionar un determinado problema. En los 1960s y 1970s, John Holland con sus estudiantes y algunos colegas de la Universidad de Michigan, inventaron el Algoritmo Genético, AG.

Holland estudió la adaptación tal como se presenta en la naturaleza y desarrolló la forma para llevar esos mecanismos al computador. En su libro "Natural and Artificial Systems, an Introductory Analysis with Applications to Biology, Control and Artificial Intelligence", 1975, presenta el AG como una abstracción de la evolución natural y presenta una teoría sobre la adaptación usando AGs.

El AG, es un algoritmo de optimización por búsqueda estocástica, basado en los mecanismos de la selección natural, que trabaja muchas posibles soluciones en forma *paralela*, que se conocen como cromosomas. Dentro de un conjunto de cromosomas, siempre se va a encontrar que algunos de ellos son mejores que otros en la solución de un problema específico. Entonces, los mejores cromosomas, los más aptos en ese ambiente problema, se seleccionen como padres para la siguiente generación. Luego, a través de un intercambio de información estructurado y aleatorio, que imita los procesos de la evolución biológica, se obtiene la siguiente generación que, en principio, va a solucionar mejor el problema. El problema que se pretende solucionar con el AG se presenta como una función de aptitud que debe satisfacer una o varias condiciones.

La estructura de datos del cromosoma, es una lista, donde se contemplan todas las posibles soluciones del problema, es decir, define el espacio de búsqueda del AG. Un cromosoma es un punto en el espacio de búsqueda de soluciones, es una solución candidata que puede ser la solución del problema. El problema que se pretende solucionar, con todas sus posibles restricciones, se define como la función de aptitud.

El AG procesa la población corriente de cromosomas, luego se reemplaza esa población creando una nueva generación con base en el comportamiento de los cromosomas de la generación anterior, con respecto a la función de aptitud. Así avanzan las generaciones. La aptitud de cada cromosoma depende de qué tan bien la solución que representa resuelve el problema. En el caso del AGS, Algoritmo Genético Simple, los símbolos que almacena la lista son 0s y 1s.

3.2 El Algoritmo Genético Simple, AGS

Los mecanismos de un AG son sorprendentemente sencillos, las operaciones más complejas incluyen la copia de listas y el intercambio parcial del contenido de las listas. La simplicidad de operación y su efectividad son dos de las principales atracciones de los AGs.

Con el fin de tener mayor claridad sobre los términos empleados, veamos algunos nombres tomados de la biología, que a veces se utilizan en el contexto de los AGs:

Biología	**Alg. Genéticos**
Cromosoma	Lista
Gen	Elemento de la lista
Genotipo	Solución codificada, cromosoma
Fenotipo	Solución decodificada

Definamos algunos parámetros de los AGs.
$\boldsymbol{l}$: número de genes del cromosoma; en AGS, número de bits o longitud del cromosoma
$\boldsymbol{k}$: un gen cualquiera del cromosoma
$\boldsymbol{K}$: número de cromosomas en la población
$\boldsymbol{M}$: número de generaciones del algoritmo
$\boldsymbol{x}$: el valor que representa un cromosoma; el cromosoma decodificado
$\boldsymbol{f(x)}$: función de aptitud aplicada en el punto $\boldsymbol{x}$.

Para trabajar con el Algoritmo Genético se han desarrollado varios operadores, sin embargo, los más importantes y comunes, que producen buenos resultados son los siguientes tres operadores:

1. Selección.
2. Cruce.
3. Mutación.

Selección. O reproducción, es un proceso en que ciertos cromosomas se llevan a la siguiente generación, considerando su comportamiento con respecto a la función de aptitud f y con respecto al resto de la población, para que se seleccionen como padres de la siguiente generación. Esta selección se hace con base en el valor *promedio de aptitud*, dándole a los cromosomas con mayor aptitud, mayor probabilidad de selección, para contribuir con uno o más descendientes en la siguiente generación. Este operador es una versión artificial de selección natural. Este operador se aplica a todos los cromosomas, la función de aptitud es el árbitro final de la vida o la muerte de los cromosomas.

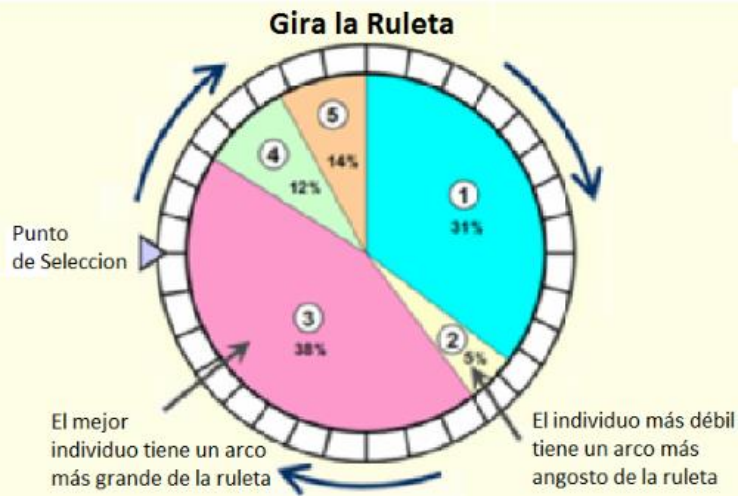


Figura 3.2 Selección por ruleta

Una forma básica de selección de los cromosomas más aptos como padres de la siguiente generación es la selección por ruleta. Como se puede observar de la figura 3.2, a los cromosomas más aptos se les asigna un arco más amplio que a los cromosomas menos aptos.

Cruce simple o cruzamiento, procede en dos pasos. **Primero,** los cromosomas seleccionados como padres entran en el juego del apareamiento, mezclándose al azar. **Segundo**, se efectúa el cruce al azar de cada par de listas como sigue: se selecciona con probabilidad uniforme, una posición $\boldsymbol{k}$, entre $\mathbf{1}$ y la longitud $\boldsymbol{l}$ de la lista menos **1,** $[\mathbf{1}, \boldsymbol{l} - \mathbf{1}]$. Se crean dos nuevas listas entre las posiciones

$\boldsymbol{k+1}$ y $\boldsymbol{l}$ incluidas. Por ejemplo, considere las listas con $\boldsymbol{l} = 5$, y los cromosomas A1 y A2, que se han seleccionado como padres:

A1 = 0 1 0 1 1 1 0 1 0
A2 = 0 1 1 1 1 0 0 1 0

Suponga que se hizo la escogencia de un número aleatorio entre 1 y 8, y se obtuvo $\boldsymbol{k}$ = 5. El cruce resultante da dos nuevas cadenas A1' y A2' que entran a formar parte de la nueva generación:

A1' = 0 1 0 1 1 0 0 1 0
A2' = 0 1 1 1 1 1 0 1 0

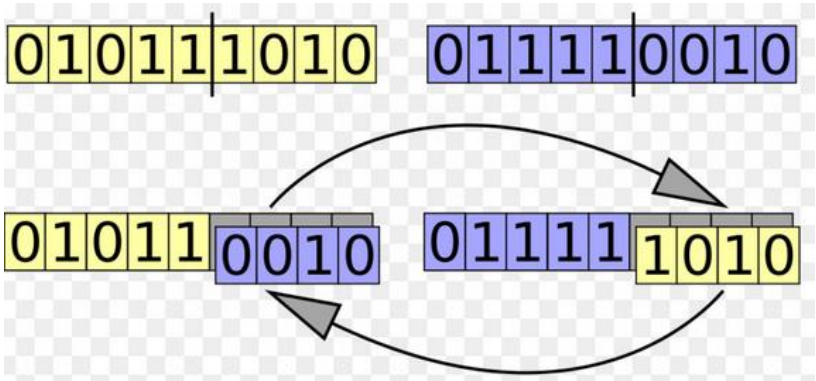


Figura 3.3. Cruce simple con $\boldsymbol{l} = 9\ y\ k = 5$

Este operador permite combinar la información de cromosomas exitosos, para la producción de descendencia, con información que se supone más exitosa que la de sus padres, lo que permite la explotación de información genética validada en determinadas partes del espacio de búsqueda.

Los mecanismos de reproducción y cruce son muy simples: generación de números aleatorios, copia de listas, e intercambio parcial de contenido de listas. Esta combinación de reproducción y de intercambio estructurado de información, aunque al azar, es lo que le da a los AGs mucha de su poder.

Mutación, es un operador genético de segundo orden, pues es mucho más provechoso el intercambio de información de cromosomas exitosos en el cruce. La mutación permite que se exploren otras áreas del espacio de soluciones, que puedan tener picos de optimización más altos. En el caso del AGS, con base en una probabilidad, se cambia un 1 por un 0, o viceversa. Este operador se aplica a los individuos con una probabilidad $\boldsymbol{p_m}$ generalmente baja.

La mutación evita que la solución se quede en un óptimo local, al explorar otras áreas del espacio de soluciones. Otra forma de verla es que aumenta la diversidad genética de la población, posibilitando mejores soluciones. En muchos casos se usa la mutación como función del tiempo (generación) de evolución, dando mayor probabilidad de mutación al comienzo del proceso. La repetición permanentemente de estos tres operadores a través de las generaciones, continúa hasta llegar o a un número determinado de generaciones o cuando

se satisfaga un criterio de finalización; tomando la mejor solución hasta ese momento como la respuesta al problema.

3.3 Algoritmo Genético Simple

A continuación, se presenta una descripción del AGS.

1. Se genera aleatoriamente la primera población. Se crean $\boldsymbol{K}$ cromosomas cada uno con $\boldsymbol{l}$ bits de longitud.
2. Se repiten $\boldsymbol{M}$ veces los siguientes pasos, o el número de veces que sea necesario hasta satisfacer otra condición de terminación del algoritmo:
    i. Se obtiene el valor $\boldsymbol{x}$, por decodificación del cromosoma..
    ii. Se calcula la aptitud $\boldsymbol{f}(\boldsymbol{x})$, de cada cromosoma; se encuentra la aptitud global de la población, que es la sumatoria de las $\boldsymbol{f}(\boldsymbol{x})$; se calcula la aptitud con que contribuye cada cromosoma a la aptitud global de la población, $\boldsymbol{f}(\boldsymbol{x})$ dividida por la aptitud global de la población, esta se toma como la probabilidad de selección de ese cromosoma.
    iii. Para cada cromosoma de la población actual se efectúa un proceso de selección con base en su probabilidad de selección, que define si tendrá o no descendientes en la siguiente generación. La selección se hace sin reemplazo, es decir, que un cromosoma puede seleccionarse más de una vez para ser padre. Se mantiene el mismo tamaño de la población.
    iv. Se escoge un par de cromosomas de la población de padres. Se cruzan el par de cromosomas en un punto escogido aleatoriamente (con una probabilidad uniforme) para formar dos hijos.
    v. De acuerdo con una probabilidad de mutación $\boldsymbol{p_m}$, se elige sustituir un gen con valor 1 por un gen con valor 0, o viceversa. Una vez aplicados estos operadores se tiene la nueva población.
    vi. Se reemplaza la población actual por la nueva población. Se disminuye en 1, el conteo de generaciones.
    vii. Regresa al paso 2.

Como la aleatoriedad juega un papel importante en la ejecución del AG, dos ejecuciones, pueden producir comportamientos diferentes en las primeras generaciones, para hacia al final converger a las mismas respuesta o a respuestas muy cercanas. Los reportes estadísticos, como aptitud promedio de la población, son en general iguales para diferentes ejecuciones del AG sobre el mismo problema.

3.4 Ejemplo de un Algoritmo Genético Simple

Supongamos que se quiere encontrar la raíz de un polinomio por AGS. Dado el polinomio $\boldsymbol{P}(\boldsymbol{x}) = \boldsymbol{x^2 - 1.23x - 6.26}$, encontrar su raíz positiva que se encuentra entre *1* y *4.*

Teniendo en cuenta que necesitamos encontrar un número entre 1 y 4, con tres decimales, el cromosoma debe almacenar ceros y unos que representen ese intervalo, estos son los datos para escoger el número ***l*** de bits de la lista. Supongamos ***l*** *= 6,* los valores que se pueden representar son:

0	0	0	0	0	0	= 0

1	1	1	1	1	1	= 63

Dado que el límite inferior es 1, la evaluación binaria de la primera lista es 0, de manera que el 1 del problema se representa por 0, el límite superior del intervalo es 4, pero dado que el 0 representa 1, entonces el 4 del problema, es 1 + 3, así el 3 se representa por 63. Lo primero que calculamos es que dec es el valor decimal del cromosoma, entonces $x = 1 + 3 * \frac{dec}{63}$. Esta es la forma en que se decodifica la solución $\boldsymbol{x}$ del problema de la estructura lista, para este ejemplo específico.

Ahora vamos a generar aleatoriamente 8 listas. Para decodificar los valores de cada lista, primero encontramos su valor decimal equivalente ***D*** al binario almacenado y luego lo transformamos a un valor entre 0 y 4, son los valores ***x***, con que vamos a evaluar la función de aptitud $\boldsymbol{P(x) = x^2 - 1.23x - 6.26}$. Entonces, evaluamos el polinomio con esos valores y miramos su resultado. En este caso definimos la aptitud como: $Aptitud = 10 - Valor\ absoluto(f(x))$.

A continuación, se desarrolla el ejemplo en Excel. Para crear la primera generación usamos la función **ALEATORIO.ENTRE(0,1).** Para encontrar la aptitud usamos la función **ABS( ).**

Pob. Nueva						***Dec***	***x***	***x2-1.123x-6.26***	***Aptitud***	***Probab.***		***Selección***							***Cruce***							***Mutación***					
0	0	1	0	0	1	9	1,429	-5,823469388	4,17653	0,0845		1	0	1	1	1	0		1	0	1	1	1	0		1	0	1	1	1	0
1	0	1	1	1	0	46	3,19	0,33623356	9,66377	0,1956		1	1	1	1	1	0		1	1	1	1	1	0		1	1	1	1	1	0
1	1	1	1	1	0	62	3,952	4,922791383	5,07721	0,1028		1	0	0	1	1	0		1	0	1	0	1	1		1	0	1	0	1	1
0	0	1	1	1	1	15	1,714	-5,246367347	4,75363	0,0962		1	0	1	0	1	1		1	0	1	1	1	0		1	0	1	1	1	0
1	0	0	1	1	0	38	2,81	-1,521671202	8,47833	0,1716		1	0	1	1	1	0		1	0	1	1	1	1		1	0	1	1	1	1
1	0	1	0	1	1	43	3,048	-0,394494331	9,60551	0,1944		1	0	1	0	1	1		1	0	1	0	1	0		1	0	1	0	1	0
0	0	0	0	1	1	3	1,143	-6,237306122	3,76269	0,0762		1	0	0	1	1	0		1	0	0	1	1	1		1	0	0	1	0	1
0	0	0	1	0	1	5	1,238	-6,117501134	3,8825	0,0786		1	0	1	0	1	1		1	0	1	0	1	0		1	0	1	0	1	0
								Aptitud total =	49,4002																						

En este caso definimos la aptitud como: $Aptitud = 10 - Valor\ absoluto(f(x))$

Se obtiene la aptitud de cada lista y la aptitud total de la población. La probabilidad de seleccionar una lista como padre de la siguiente generación, viene dada por la relación entre la aptitud propia de cada lista y la aptitud total de la población. A través de varios mecanismos, como puede ser la ruleta, se seleccionan las listas padre para la nueva población. Es común que la mejor lista se repita varias veces en la población de listas seleccionadas, en este caso la lista 101110. Se selecciona dos veces. Luego, el cruce, se toman dos listas seguidas y a partir de una posición aleatoria se intercambian los contenidos. Posteriormente, la mutación, se selecciona una posición para cambiar un 1 por 0, o viceversa. Se obtiene la nueva población.

Pob. Nueva						Dec	x	x2-1.123x-6.26	Aptitud	Probab.	Selección						Cruce						Mutación					
1	0	1	1	1	0	46	3,19	0,33623356	9,66377	0,1372	1	0	1	1	1	0	1	0	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	0	62	3,952	4,922791383	5,07721	0,0721	1	0	1	0	1	1	1	0	1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	1	43	3,048	-0,394494331	9,60551	0,1364	1	0	1	1	1	0	1	0	1	1	1	1	1	0	1	1	0	1
1	0	1	1	1	0	46	3,19	0,33623356	9,66377	0,1372	1	0	1	1	1	1	1	0	1	1	1	0	1	0	1	1	1	0
1	0	1	1	1	1	47	3,238	0,588879819	9,41112	0,1336	1	0	1	0	1	0	1	0	1	1	1	0	1	0	1	1	1	0
1	0	1	0	1	0	42	3	-0,629	9,371	0,1331	1	0	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0
1	0	0	1	0	1	37	2,762	-1,733501134	8,2665	0,1174	1	0	1	0	1	1	1	0	1	1	1	0	1	0	1	1	1	0
1	0	1	0	1	0	42	3	-0,629	9,371	0,1331	1	0	1	1	1	0	1	0	1	0	1	1	1	0	1	0	1	1
								Aptitud total =	70,4299																			

Ahora se tiene la nueva población y se repiten los mismos pasos. Aquí la aptitud global aumenta más de un 57%, con respecto a la generación anterior. Visto de otra manera, es más difícil ser competitivo en esta población. Es importante considerar que solo estamos utilizando 6 bits, por lo que tenemos muy poca resolución en la respuesta. También es importante observar que las mejores listas tienen

1	0	1

en sus tres primeras posiciones, este es un esquema exitoso. Así tenemos que cualquiera sea la solución del problema la lista más apta debe llevar esos tres bits con esos valores. El teorema del Algoritmo Genético dice que las mejores soluciones se componen de esquemas exitosos y lo que hace la evolución es conformar la mejor solución a partir de esquemas exitosos.

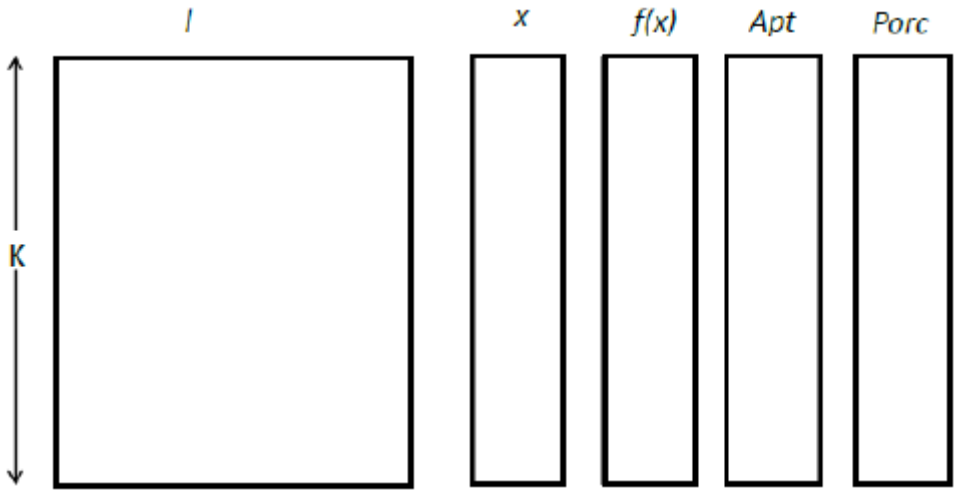
Pob. Nueva						Dec	x	x2-1.123x-6.26	Aptitud
1	0	1	1	1	1	47	3,238	0,588879819	9,41112
1	0	1	0	1	0	42	3	-0,629	9,371
1	0	1	1	0	1	45	3,143	0,088122449	9,91188
1	0	1	1	1	0	46	3,19	0,33623356	9,66377
1	0	1	1	1	0	46	3,19	0,33623356	9,66377
1	0	1	0	1	0	42	3	-0,629	9,371
1	0	1	1	1	0	46	3,19	0,33623356	9,66377
1	0	1	0	1	1	43	3,048	-0,394494331	9,60551
								Aptitud total =	76,6618

Ahora con la nueva población repetimos el proceso, pero lo detenemos, al encontrar que la tercera lista de la población nos da la solución del problema. Este es uno de los criterios de detención del A.G., el otro criterio, el más común es cuando el A.G. se ha corrido un número de generaciones definido previamente.

3.5 Codificación de las Rutinas Básicas de un Algoritmo Genético

Rutina principal

Corresponde a la descripción que acabamos de estudiar. La estructura de datos que se va a utilizar es la de una matriz de bits $\boldsymbol{l} * \boldsymbol{K}$, y cuatro vectores asociados; $\boldsymbol{x}$, que es el resultado de decodificar cada cromosoma, $\boldsymbol{f}(\boldsymbol{x})$ que es la evaluación de la función de aptitud, $\boldsymbol{Apt}$, que es la aptitud de cada cromosoma, y , $\boldsymbol{Porc}$ que es el porcentaje o probabilidad con que ese cromosoma contribuye a la aptitud total de la población.



El ciclo principal de la rutina se ejecuta $\boldsymbol{M-1}$ veces. La variable $\boldsymbol{i}$ se emplea como índice de las iteraciones. La estructura pob guarda los $\boldsymbol{K}$ cromosomas de longitud $\boldsymbol{l}$ que se generan inicialmente.

```
def Alg_Genetico(ecuacion,l, M, K, p_mut):
   pob = genera(l,K)  #Se genera aleatoriamente la población
   probab = eval_apt(pob,l) #Se evalúa esa primera población
   i =0;                #Es la primera generación
   while i < M:     #Mientras no se ejecuten todas las generaciones
      n_pob = seleccion(pob, probab) #Selecciona, los nuevo padres
      hijos = cruce(n_pob)          #Cruce de los padres generan hijos
      pob = mutacion(hijos, p_mut, l)   #Se mutan los hijos para generar otra población
      probab = eval_apt(pob,l) #Se evalua la población i, se calcula la prob. de selección
      i = i + 1
```

Rutina Genera

Esta rutina simula el lanzamiento de una moneda, para generar aleatoriamente un bit.

```
def genera(l,K):
   pob = [randint(0, 2, l).tolist() for _ in range(K)]
   return pob
```

Rutina evalúa

Una vez se tiene la primera generación, se debe evaluar el comportamiento de cada cromosoma con respecto a la función de aptitud. Luego encontrar cuál es su aporte a la aptitud global de la población para saber qué tan bueno es con respecto a los demás.

La aptitud total de la población se guarda en $\boldsymbol{apt_pob}$, la aptitud del cromosoma $\boldsymbol{i}$ que guarda en el vector asociado apt_crom. La aptitud con que contribuye el cromosoma $\boldsymbol{i}$ a la aptitud global, en porcentaje se guarda en el vector asociado $\boldsymbol{probab}$.

```
def eval_apt(pob,l):
   apt_crom = []
```

```
    apt_pob = 0
    for i in range(0, len(pop)):
        crom = pop[i]
        apt = evalua(crom,l)
        apt_pob += apt
        apt_crom.append(apt)
    probab = []
    for i in range(0, len(pop)):
        prob_crom = apt_crom[i]/apt_pob
        probab.append(prob_crom)
    return probab
```

Función F_aptitud

Esta función, que devuelve un valor real, es la encargada de evaluar la aptitud del cromosoma $\boldsymbol{i}$ de la rutina anterior. La decodificación del cromosoma $\boldsymbol{i}$ produce el valor $\boldsymbol{x}$ con que se evalúa esta función. Para ilustrar mejor el algoritmo, tomamos como ejemplo maximizar una función real arbitraria: $\boldsymbol{f}(\boldsymbol{x}) \ == \ 5 * x ** 5 - 3 * x ** 4 - x ** 3 - 5 * x ** 2 - x - 3$. Se llama a la función ***decodifica,*** que convierte la cadena binaria ***cromosoma*** en un valor del espacio de soluciones.

```
def evalua(crom,l):
    x = decodifica2(crom,l)
    y = ecuacion(x)
    apt = 15 - abs(y)
    return apt

def ecuacion(x):
    valor = 5*x**5-3*x**4-x**3-5*x**2-x-3
    return valor
```

Función Decodifica

Esta función convierte el contenido binario del cromosoma, el genotipo, en un valor real, el fenotipo, dentro del espacio de soluciones. Cómo existen diversas formas de representar el espacio de soluciones en la estructura del cromosoma, para cada una de esas formas hay un esquema propio de codificación.

La función ***decodifica*** es la que realmente hace que el AG sea universal, ya que con solo modificarla se pueden representar una amplia variedad de problemas. La variable ***rango*** define el intervalo en el cual puede estar el valor real del cromosoma, para el ejemplo rango es $[0.5,\ 1.5]$. En ***valorDecimal*** se calcula el valor del cromosoma en binario.

```
def decodifica(crom,l):
    xi = 0.5  # Limite inferior en el dominio real
    xf = 1.5  # Limite superior en el dominio real
    Max = 2 **(l)
    cromPot = [crom[i]*2**(l-i-1) for i in range(l)]
    #print(cromPot)
    valorDecimal = sum(cromPot)
    #print(valorDecimal)
```

```
    valDeco = ((xf-xi)/Max)*(valorDecimal)+xi
    return valDeco
```

Procedimiento de selección

Este procedimiento es la versión artificial de la selección natural, que busca un superviviente darwiniano entre los cromosomas más aptos; en las poblaciones naturales la aptitud está definida por la habilidad del individuo para sobrevivir a predadores, pestes y otras situaciones que reducen la posibilidad de que el individuo llegue a adulto y pueda reproducirse. El ciclo externo se debe a que se trabaja con población constante y se deben seleccionar $\boldsymbol{K}$ individuos. El ciclo interno es la búsqueda de los mejores individuos dentro de la población, que se debe repetir hasta completar los $\boldsymbol{K}$ individuos, es por esto los mejores individuos se seleccionan varias veces.

```
def seleccion(pob, probab):
    j = 0
    K = len(pob)
    #print("long pob", K)
    #k = len(probab)
    #print("long probab", k)
    limite = 2 * max(probab)
    pob_nueva = []
    while j < K:
        i = 0
        while i < K:
            aleat = random.uniform(0, limite)
            if probab[i] > aleat:
                pob_nueva.append(pob[j])
                j += 1
                if j >= K:
                    break
            i +=1
            if i == K:
                i = 0
    return pob_nueva
```

Procedimiento de Cruce

Este procedimiento no utiliza la probabilidad de cruce, se implementa fácilmente considerando que la población se ha ido generando aleatoriamente. Simplemente se seleccionan dos parejas consecutivas de la población seleccionada y se cruzan en un punto escogido aleatoriamente. ***pt*** es el punto para el cruce y es una posición aleatoria entre $\boldsymbol{2}$ y $\boldsymbol{l} - \mathbf{1}$. Este es el cruce más elemental, se pueden hacer cruces multipunto, lo que mejora el comportamiento del algoritmo pues elimina sesgos en la población.

```
def cruce(pob_nueva):
    i = 0
    K = len(pob_nueva)
    hijos =[]
```

```
  while i < K:
    crom1 = pob_nueva[i]
    crom2 = pob_nueva[i+1]
    pt = randint(1, len(crom1)-1)
    hijo1 = crom1[:pt] + crom2[pt:]
    hijo2 = crom2[:pt] + crom1[pt:]
    hijos.append(hijo1)
    hijos.append(hijo2)
    i = i +2;
  return(hijos)
```

Procedimiento de Mutación

Considerando que la probabilidad de mutación en general es muy baja y con el fin de ahorrar tiempo de proceso, se toma su inverso como un bloque o segmento en el cual debiera haber una mutación. Se encuentra el número de mutaciones y dentro del bloque se lanza ese número de aleatorios para cambiar en las posiciones dadas.

```
def mutacion(hijos, p_mut, l):
  K = len(hijos)
  totalbits = K * l
  segmento = 1/p_mut
  n_mutaciones = totalbits/segmento
  i = 0
  while i < n_mutaciones:
    muta = randint(0, totalbits-1)
    x = math.floor(muta/l)
    y = muta%l
    if hijos[x][y] == 0:
      hijos[x][y] = 1
    else:
      hijos[x][y] = 0
    i += 1
  return hijos
```

Otras funciones

Dependiendo del nivel de sofisticación del programa, se pueden incluir rutinas para la impresión de los cromosomas, sus aptitudes, graficas del mejor cromosoma por generación, y otros; según las necesidades de análisis del algoritmo. Por ejemplo, llevar el mejor cromosoma y su aptitud en cada generación, permite observar cómo es la convergencia del algoritmo. En caso de no llegar a una convergencia, se puede replantear la codificación de las variables, o la evaluación de la función, o el método de selección o la estructura del cromosoma.

De lo anterior se puede ver la potencialidad del AG, que a partir de un proceso aleatorio se van generando soluciones, en este caso, para funciones algebraicas. Sin embargo, el AG es una herramienta que no se limita a usar genes binarios, sino que se puede utilizar otros tipos de datos para la solución combinatoria de cualquier problema, como se mencionó.

En las aplicaciones que actualmente se trabajan, dependiendo del problema, se tienen diferentes tipos de datos para los genes, lo que lleva a variar y aumentar los operadores de cruce y mutación; aunque también es común la creación de nuevos operadores genéticos.

3.6 Problemas Tipo NP y NP-Completos

3.6.1 Problemas Tipo P

Un algoritmo cuyo tiempo de ejecución está dado por un polinomio que es función del tamaño de los datos de entrada, se conoce como un algoritmo con tiempo de ejecución polinomial. Un problema pertenece a la clase $\boldsymbol{P}$ si el número de pasos requeridos para hallar su solución, es decir, su tiempo de ejecución, está limitado o puede definirse por un polinomio, el problema se conoce como de Tipo P. Este tipo de problemas evita preocupaciones en cuanto al modelo de máquina, dado que todos los modelos de computación, incluyendo la máquina de Turing, manejan complejidad de tiempo relacionada polinomialmente. En su gran mayoría son los problemas que se trabajan permanentemente.

3.6.2 Problemas Tipo NP

Los problemas $\boldsymbol{NP}$ son intratables en el sentido de que no se ha encontrado una solución real y eficiente para el problema, cuando el tamaño de los datos de entrada está dado por un $\boldsymbol{n}$ grande. En este caso, el tiempo necesario para hallar una solución puede tener un crecimiento exponencial o aún mayor con $\boldsymbol{n}$. Un problema pertenece a la clase de problemas de tiempo polinomial no determinístico, $\boldsymbol{NP}$, si se puede resolver en tiempo polinomial en una máquina de Turing no determinística. Una máquina de Turing no determinística $\boldsymbol{MTN}$, es una máquina de Turing paralela que puede tomar muchos caminos computacionales simultáneos, con la restricción de que las máquinas de Turing paralelas no se pueden comunicar.

Algunos algoritmos han podido encontrar buenas soluciones a problemas **NP**, sin embargo, no se puede afirmar que se haya encontrado la solución óptima. Por ejemplo, el problema de asignar los horarios de los exámenes finales, en una universidad con $\boldsymbol{n}$ cursos y con solo 5 días para efectuar los exámenes. Otro es el bien conocido problema de encontrar el recorrido completo de un agente viajero, que debe visitar $\boldsymbol{n}$ ciudades tan solo una vez y al final llegar a la ciudad de partida, en un tiempo mínimo.

3.6.3 Problemas Tipo NP Completos

Los problemas NP conocidos, tienen una característica importante: todos se pueden reducir a uno. Es decir que, dados dos problemas NP, X y Y, existe un algoritmo de tiempo polinomial que redefine un problema de tipo X como un problema de tipo Y, y existe un algoritmo de tiempo polinomial que traslada una solución de un problema de tipo Y a una solución para un problema de tipo X. O sea, que, si se encuentra un algoritmo de tiempo polinomial para cualquiera de estos problemas, entonces, hay un algoritmo derivado, de tiempo polinomial, para todos los demás problemas del conjunto. Un problema que satisface estos requerimientos se denomina un problema NP - Completo.

Por ejemplo, el problema del agente viajero es un problema NP-Completo, así que cualquier otro problema que se pueda transformar en este problema también es un problema NP-Completo, que es lo que en general sucede con los problemas de asignación. Otro ejemplo es el problema del transporte, en el que se debe determinar un conjunto de rutas que debe seguir un conjunto de vehículos que parten de uno o más depósitos para satisfacer la demanda de varios clientes dispersos geográficamente.

Ha habido mucho trabajo entre los matemáticos alrededor de si un problema tipo P es igual a un problema tipo NP, pues esto llevaría a la solución en tiempo polinomial de muchos problemas tipo NP, sin embargo en el artículo de D. Cardona, se demuestra que P ≠ NP, utilizando ACs.

3.7 El Problema del Furgón o de la Mochila

Para comprender como trabaja el AG en estos problemas, consideremos un ejemplo concreto, el problema de la mochila, que es un problema NP-Completo clásico. Estableciéndolo simplemente es: hay una colección de artículos que varían en peso y valor, que se quieren cargar en una mochila que tiene una determinada capacidad, se debe encontrar una combinación de artículos para llevar, que tenga la mayor utilidad total y que no exceda un peso límite.

En otras palabras, la meta está en llenar completamente una mochila hipotética con el lote más útil que puede llevar. Aunque es fácil de describir, esta meta puede ser difícil de realizar. Por ejemplo, una colección de solo 50 artículos presenta 2^{50} posibles combinaciones diferentes. Suponiendo que un computador pueda probar un millón de combinaciones diferentes por segundo, tomaría alrededor de 35 años probarlas todas.

Aquí se muestra cómo un AG resuelve tal problema, pero en aras de la ilustración, se usa un número mínimo de artículos. La colección contiene 8 artículos, así que hay 256 combinaciones posibles, para cargar la mochila. Cada artículo tiene un peso y un valor de utilidad, ver la Tabla 3.1. El furgón puede cargar un peso máximo de 25 unidades, así que puede llevar un A, o dos B's, etc. La Tabla 1 enumera todos los artículos, las unidades con sus pesos y valores; y el número disponible de cada tipo de artículo para transportar.

Tabla 3.1

Artículo	Descripción	Peso (kg)	Valor (utilidad)
B	Botella de agua	3	8
L	Linterna	2	7
K	Kit de primeros auxilios	4	11
R	Ropa extra	5	6
C	Comida enlatada	7	16
S	Saco de dormir	6	12
M	Mapa y brújula	1	7
Cu	Cuerda de escalada	8	18

Aprovechando dar solución a este problema, a continuación, se presentan los pasos que se siguen para solucionar un problema por AG.

3.8 Diseño de un AG para la solución de problemas

Como ya se ha dicho, la estructura general de AG es común para la solución de problemas diferentes independiente de su dominio. Para diseñar la solución de un problema con AGs, se deben seguir los siguientes pasos:

1. Definir el problema. Debe haber claridad con respecto al problema que se pretende solucionar por algoritmos genéticos, en términos de complejidad computacional.
2. Definir la representación de los individuos, o diseñar los genes que conforman el cromosoma. Define el espacio de búsqueda de las todas las soluciones.
3. Definir los operadores genéticos. Buscar los operadores apropiados de selección, cruce y mutación, o crearlos.
4. Definir la función de aptitud. Nos permite medir el desempeño de cada individuo. En algunos casos es posible que haya que incluir penalidades.
5. Definir el criterio de parada y la población.
6. Definir otros parámetros. Como la población y la probabilidad de mutación.

Es importante aclarar que estos pasos son una guía, dado que en cualquier momento, dependiendo del comportamiento del algoritmo genético, se debe volver a algunos pasos anteriores para afinar ese comportamiento.

1. Definir el problema. El algoritmo genético es una técnica de optimización, en ese sentido se debe estudiar el problema: primero, debe tener claridad en que se lo que se quiere optimizar; segundo, que restricciones debe satisfacer; y tercero, cual es el tipo de solución, numérica o una lista de elementos.

2. Definir la representación de los individuos. El diseño del individuo tiene que ver con la forma en que se van a expresar las soluciones del problema. Si se busca la solución de una ecuación a través de una variable, la solución debe ser un valor real. Si la solución es un recorrido, la solución debe ser una serie de puntos (x, y).

El diseño de cada uno de los genes que conforman el cromosoma, define el tipo de datos de cada gen y su forma de codificación, que naturalmente dependen del dominio del problema que se va a solucionar.

El diseño del cromosoma, o la estructura de datos, que determina cómo representar una solución potencial del problema planteado, define el espacio de soluciones del problema, en términos de la codificación de los genes que la componen. En algunos casos también se debe tener en cuenta que la longitud del cromosoma puede ser más larga o menos, al considerar la precisión que se le quiera dar a su solución, como en el caso de encontrar la raíz de una ecuación.

3. Definir los operadores genéticos. Los tipos de datos de los genes pueden ser binarios, enteros, reales, lógicos, o por extensión cualquier tipo de objeto. Nos dan la pauta para la selección o creación de los operadores apropiados. Dependiendo del dominio del problema, se deben definir y adecuar los operadores genéticos que se van a utilizar en el AG.

Generalmente, más que definir los operadores genéticos apropiados es escogerlos dentro de una gama de operadores bien conocidos. Los más comunes son los operadores binarios, aunque también en ellos se encuentra una gama de posibilidades. En el caso de los operadores de tipo real, aparecen diferentes opciones, dependiendo del dominio del problema a solucionar. Pero también puede uno pensar en objetos y definir sus propios operadores genéticos. Supongamos que estoy trabajando con voz; podría definir una clase con objetos voz y sus operadores genéticos.

4. Definir la función de aptitud. Permite decidir y medir la bondad de la respuesta del AG, es una de las consideraciones más importantes a tener en cuenta en el diseño del AG, pues de ella depende todo el comportamiento del AG.

En muchos casos en que el problema tiene restricciones, como en el Problema del Furgón, se presenta que algunos cromosomas violan las restricciones. Los investigadores de AG han explorado muchos enfoques a la violación de restricciones, pero ninguno es perfecto. Aquí hay tres posibilidades:

> **Eliminación**. La eliminación intenta determinar si un cromosoma infringe la restricción, aún antes de su creación. Este enfoque tiene varios problemas. En primer lugar, puede ser demasiado costoso en su ejecución, o simplemente imposible. Segundo, al prevenir la creación de cromosomas que infrinjan las restricciones, se puede ocasionar que el AG pase por alto soluciones perfectamente válidas. Esto porque los infractores podrían producir descendientes legales (no infractores) que llevarían a una solución satisfactoria más rápidamente.
>
> **Alta Penalidad.**
> Este enfoque impone una pena alta a los infractores. Reduce la aptitud de los infractores, permitiéndoles, ocasionalmente, propagar su descendencia. Una debilidad de este enfoque se hace evidente, cuando una población contiene gran porcentaje de infractores. En este caso, los cromosomas legales dominarán las generaciones siguientes y los infractores se dejarán sin explotar. Este efecto podría conducir al estancamiento de la población.
>
> **Penalidad Moderada.** Un tercer enfoque impone una penalidad moderada a los infractores. Aumenta la probabilidad de que los infractores procreen, reduciendo así la oportunidad de que la población se estanque. También presenta sus propios problemas, especialmente cuando los infractores se clasifican mejor que los cromosomas legales. En este caso, si los infractores no crean listas legales, entonces los infractores dominarán las generaciones siguientes. Además, si los infractores clasifican más alto que las listas legales, entonces los criterios para terminar la búsqueda deben incorporar un mecanismo para detectar infractores.

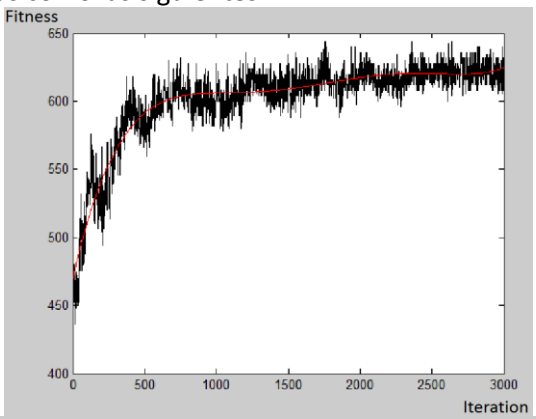
Con respecto a la población, el número de cromosomas es proporcional a la complejidad del problema. Esto puede conllevar gastos fuertes en poder de cómputo, pues el proceso de cada cromosoma requiere más cálculos. Este es uno de los defectos del AG, ya que después de procesar, en cada generación $\boldsymbol{K}$, cromosomas, al final solo se requiere un cromosoma como respuesta. Así la población debe ser lo suficientemente grande para crear un conjunto diverso de individuos, para que el AG pueda explotar mejor el espacio de búsqueda. El número de generaciones tiene que ver con la población, a mayor población, mayor número de generaciones.

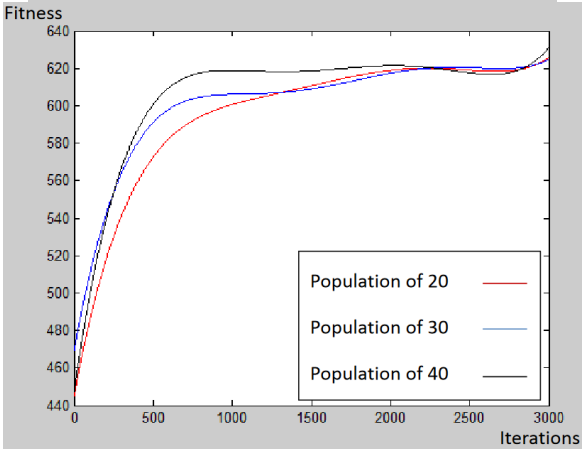
5. Definir el criterio de parada. El criterio de parada más sencillo hace referencia al número de generaciones del AG, pues en muchas ocasiones no es fácil establecerlo precisamente en términos de la función de aptitud. La idea es que el AG se va afinando para incluir un criterio de parada en términos de la solución.

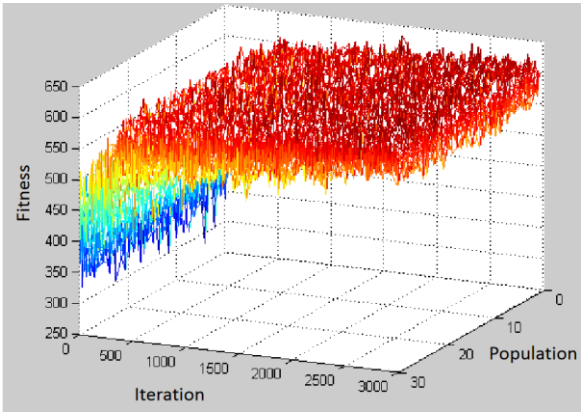
Estos pasos, como los de cualquier otro diseño en ingeniería, si bien se plantean secuencialmente, son de un ir y volver, hasta llegar a un mejor diseño. La combinación apropiada de estas tareas, le da al algoritmo genético la potencialidad para resolver el problema.

En el caso de problema NP-Completos, el criterio de parada más común es el número de generaciones, dado que la explosión combinatoria es tan alta que no se puede llegar a conocer la respuesta o configuración óptima.

7. **Definir otros parámetros.** Son varios los parámetros, además del número de generaciones, la población y la probabilidad de mutación, que se pueden considerar como: el mejor cromosoma de cada generación, la mejor aptitud en cada generación y la aptitud global de la población. Otra ayuda puede ser una representación gráfica del espacio de búsqueda, para poder observar cómo están distribuidas las soluciones y si falta explorar alguna parte del espacio de búsqueda, especialmente en las primeas generaciones. Con estos datos se pueden obtener graficas como las siguientes:







Con respecto a la probabilidad de mutación, como decíamos es muy baja, pero en algunos casos se puede hacer dependiente de la generación, disminuyendo en la medida en que aumenta el número de generaciones.

3.8 Diseño de un AG para el problema del Furgón

3.8.1 Pasos

1. Definir el problema. Dada una colección de artículos que varían en peso y valor de utilidad, según la tabla 3.1, se quiere transportar una combinación de artículos en una mochila con una capacidad de carga de máximo 24 unidades de peso, que tenga el mayor valor de utilidad total.

2. Definir la representación de los individuos. Lo que se va a evaluar en cada generación del AG es una población de cromosomas o listas con el número de elementos de cada artículo que se van a transportar. Se debe representar el número de elementos de cada artículo, que conforma la lista. De manera que el tipo de datos de cada gen puede ser: binario o entero.

Para el ejemplo vamos a usar binarios, con la aplicación de operadores genéticos conocidos. Dado que el cromosoma o estructura de datos debe permitir incluir todos artículos de la lista que se pueden llevar en la mochila. En este caso concreto es un cromosoma con la posibilidad de incluir o no un determinado artículo.

Tabla 3.2 Esquema de codificación de los artículos para la lista

Posición	1	2	3	4	5	6	7	8
Artículo	B	L	K	R	C	S	M	Cu

3. Operadores. Vamos a usar los operadores binarios ya conocidos de cruce y mutación, dado que se va a trabajar con números binarios.

4. Aptitud. En este caso una función de aptitud muy simple sería cuantificar el valor total de cada una de las listas de la población, así las listas de artículos con mayor aptitud serían las que den mayor valor transportado, es decir cargar todos los artículos en el furgón.

Sin embargo, hay que imponer la restricción del peso, así las listas de artículos con mayor peso menor que la restricción son las más aptas. ¿Pero qué pasa con las listas de artículos que pesan más? ¿Las eliminamos? ¿o las penalizamos? Su función de aptitud suma el valor de cada artículo y resta una penalidad moderada para los infractores. La penalidad es tres veces la cantidad del exceso de peso. Para esto primero se define V, el valor total de la lista y P, el peso total de la lista. Donde x_i es el número de elementos del artículo i; v_i el valor de un artículo i; y p_i el peso de un artículo i. Entonces,

$$V = \sum_{i=1}^{8} v_i \ x_i \quad \text{y} \quad P = \sum_{i=1}^{8} p_i \ x_i$$

De manera que $f_{apt}(X) = V$ si $P \leq 25$; o $f_{apt}(X) = V - 2(P - 25)$

De una primera generación aleatoria se tomaron 4 listas, como las que se muestran en la Tabla 3.3. La primera lista coloca solo 2 artículos en la mochila: un kit de primeros auxilios y una cuerda de escalada; para un peso total de 12 unidades y una utilidad total de 29 unidades. La segunda lista lleva seis artículos en la mochila: una linterna, un kit, ropa extra, comida enlatada, un saco de dormir y mapa y brújula; con un peso de 25 y utilidad de 59. La tercera lleva agua, el kit, ropa extra y comida enlatada; con un peso de 17 y utilidad de 41. La cuarta lista lleva 6 artículos: agua, linterna, ropa extra, comida enlatada, un saco de dormir y mapa y brújula; con un peso de 26 unidades y un valor de 45

Artículo	**Descripción**	**Peso (kg)**	**Valor (utilidad)**
B	Botella de agua	3	8
L	Linterna	2	7
K	Kit de primeros auxilios	4	11
R	Ropa extra	5	6
C	Comida enlatada	7	16
S	Saco de dormir	6	12
M	Mapa y brújula	1	7
Cu	Cuerda de escalada	8	18

Tabla 3.3. Listas generadas aleatoriamente, que representan los artículos cargados en el furgón

Posición	1	2	3	4	5	6	7	8	Peso	Utilidad
Artículo	B	L	K	R	C	S	M	Cu		
Lista 1	0	0	1	0	0	0	0	1	12	29
Lista 2	0	1	1	1	1	1	1	0	25	59
Lista 3	1	0	1	1	1	0	0	0	17	41
Lista 4	1	1	0	1	1	1	1	0	26	45

La Tabla 3.4 muestra la aptitud resultante de aplicar la función de aptitud a las cinco listas tomadas de ejemplo anterior.

Tabla 3.4. Listas de artículos cargados en la mochila con pesos, valores y aptitud

Listas	1	2	3	4
Pesos	12	25	17	26
Utilidad	29	59	41	45
Aptitud	29	59	41	43

5. Criterio de parada. Dado que es un problema de combinatoria, el algoritmo va a parar cuando se cumpla el número de generaciones definido.

6. Definir otros parámetros. En este caso vamos a parar cuando se hayan corrido 100 generaciones, M = 100 con una población K = 30 individuos; Se escogió esta población más bien pequeña para ilustrar mejor como trabaja el AG.

3.8.2 Iniciación.

Después de adelantar los pasos anteriores, el AG está listo para correr. La primera tarea del AG es crear una población inicial de listas. Hay muchas maneras para seleccionar una población inicial; los enfoques oscilan desde definir aleatoriamente cada carácter de cada cadena, hasta modificar interactivamente los resultados de una búsqueda hecha anteriormente.

La composición de la población inicial puede afectar dramáticamente el desempeño del algoritmo genético. La mayor diversidad en la población inicial le da más oportunidades al AG, para explotar el espacio de búsqueda. El esquema del lanzamiento de la moneda tiene la ventaja de la simplicidad y la diversidad. Es simple pues no requiere ninguna información sobre el problema. El esquema es diverso porque la función crea listas que van desde casi cero hasta uno y todos los valores entre ellos.

Aquí es de destacar el algoritmo de generación de números aleatorios, porque muchos de los generadores de números aleatorios, realmente son pseudoaleatorios, se repiten después de cierto tiempo.

3.8.3 Tipos de selección

Selección de padres y procreación

Después que se construye la población, el programa principal entra en un ciclo while, y permanece allí hasta que se encuentre una solución. Es poco probable que alguna lista de la generación inicial contenga una solución. El primer paso para crear la siguiente generación es de nuevo la selección de dos padres. El método de seleccionar los padres en un AG es muy importante ya que puede incidir significativamente en la eficiencia de la búsqueda. Hay muchos tipos de formas de seleccionar los padres para la siguiente generación y veamos algunos.

Ruleta

Es el tipo de selección más conocido y que hasta ahora se hemos utilizado. En este caso la selección de una lista determinada es proporcional a su aptitud. En primer lugar, se calcula la aptitud de cada cromosoma; con base en estos valores de calcula la aptitud global de la población. Ahora, se divide la aptitud de cada cromosoma por la aptitud global de la población y así se tiene una probabilidad de selección de esa estructura, así las mejores estructuras, tienen mayor probabilidad de que se seleccionen. Se decide si el cromosoma se selecciona como padre, generando un número aleatorio, si es menor que la probabilidad del cromosoma, ese cromosoma se selecciona, de lo contrario no.

Selección por Elitismo
Hay varias versiones de este tipo de selección, la que se presenta a continuación permite mantener la diversidad genética incluyendo siempre individuos menos aptos. Aquí se toma la población con sus cromosomas ya calificados por la función de aptitud, se ordena con base en los valores de aptitud de cada estructura. Automáticamente se selecciona un porcentaje de las mejores estructuras de población. Por ejemplo, en una población de 100 cromosomas, se selecciona directamente el 20% es decir los 20 mejores. Luego simulando el lanzamiento de una moneda, se van seleccionando uno de cada dos cromosomas de la población ordenada que queda. Así muchos cromosomas con una aptitud baja tienen una probabilidad de selección más alta que en la ruleta, lo que permite una mayor distribución inicial de las estructuras en el espacio de búsqueda.

Selección por clasificación
El propósito es impedir la convergencia demasiado rápida, ya que los mejores cromosomas no difieren mucho en su aptitud. Como en el caso anterior, los cromosomas de la población se ordenan de acuerdo con la aptitud, y se obtiene un nuevo valor de aptitud de cada cromosoma, que depende más de su clasificación que de su aptitud absoluta.

El mejor cromosoma tendrá una aptitud K, el tamaño de la población; y el último tendrá una aptitud de 1. La formación de clases evita darle mucha aptitud a los más mal clasificados, aunque tienen oportunidad de ser seleccionados, suponemos R clases. Se puede compartir mucho la descendencia, con un grupo pequeño altamente apto, y así reducir la presión de selección, cuando la varianza de la aptitud es alta. Para calcular la probabilidad de selección se calcula $SumR_i$, que es la suma de todas las aptitudes de la población por clases. $R_{i,j}$ es la clase del cromosoma j en la generación i.

$$SumR_i = \sum_{i=1}^{K} R_{i,j} \text{ y}$$

La probabilidad de selección del cromosoma j en la generacion i es:

$$p(R_{i,j}) = \frac{R_{i,j}}{SumR_i}$$

Selección ayudada por el diseñador
Sin embargo, como ya se ha dicho, estas son las formas convencionales de hacer la selección y puede haber muchas otras formas de hacerlo, combinando diferentes métodos, que en muchos casos dependen del dominio del problema; o con la participación del diseñador. En este caso se presentan dos formas.

1. Ayudada o Interacción. Aquí el diseñador va observando el desarrollo del proceso y encuentra que por alguna razón ciertos espacios del espacio de búsqueda no se han explorado, entonces directamente inserta algunos cromosomas como padres, que permitan realizar la exploración en ciertas partes del espacio de búsqueda.

2. Directa. En este caso es el diseñador quien directamente califica cada cromosoma. Para esto primero define los criterios de calificación del cromosoma dependiendo del tipo de solución que se esté trabajando. Puede ser por ejemplo en un estudio sobre imágenes, donde el ojo del diseñador define la aptitud de cada imagen de la población, en estos casos la población no debe ser muy

grande, para que el ojo no se fatigue y pueda valorar los cromosomas sin parcialidad. También se puede dar en aplicaciones de audio.

3.8.4 Cruce

Lo mismo que con el operador de Selección hay varios tipos de cruce, el común, es el cruce uniforme que ya hemos estudiado. Que tiene problemas de parcialidad ya que muchas veces los primeros y los últimos bits no se cruzan tan frecuentemente como los otros bits de la lista.

Cruce en un punto

Es el que hemos estudiado en los ejemplos anteriores. Sin embargo, tiene un par de limitaciones. Una que es muy probable que genes vecinos sigan juntos y la otra que los genes de los extremos nunca estén juntos. Este sesgo posicional puede llevar, dependiendo del problema, a una exploración pobre en el espacio de soluciones.

Cruce multipunto.

En este caso se definen aleatoriamente varios puntos donde efectuar el corte, dependiendo de la longitud del cromosoma. Por ejemplo, si se definen tres puntos, aleatoriamente se distribuyen los puntos para cada par de padres Así, si se tienen los padres:

0	0	0	1	0	0
1	0	1	1	1	1

Se seleccionan dos puntos de cruce en los padres, los hijos quedan:

0	0	1	1	1	0
1	0	0	1	0	1

3.8.6 Mutación.

Mutación por Similitud

A veces los hijos experimentan mutación. En el ejemplo clásico del furgón usa un operador de mutación interesante. En lugar de una probabilidad fija de mutación, se usa una probabilidad que cambia con base en la conformación de la población. Esta rutina compara los dos padres del hijo; la mayor similitud entre padres aumenta la probabilidad que ocurra una mutación en el hijo.

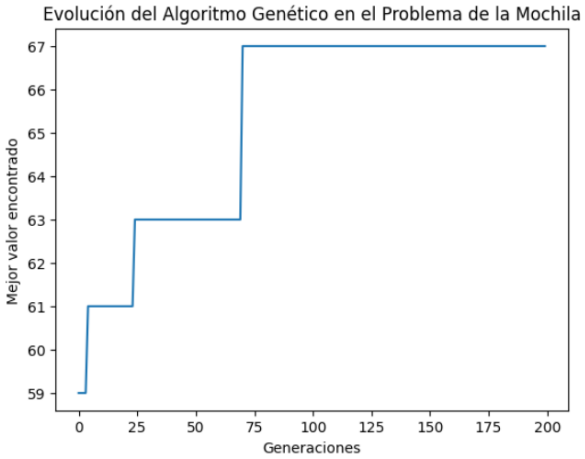
Mutación variable

Otra forma de operar la mutación es haciéndola variable en el tiempo. En este caso la probabilidad inicial de mutación es muy alta, en la medida en que pasan las generaciones la probabilidad es más baja. La razón para usar una probabilidad de mutación variable está en reducir la oportunidad de que haya convergencia prematura. Esta condición ocurre cuando la población se precipita hacia una solución mediocre y entonces simplemente se desperdician recursos. Hay poca diversidad y la búsqueda se convierte en una caminata aleatoria entre el promedio.

Encontrando la Respuesta

El AG continúa durante muchas generaciones aplicando los operadores ya definidos, pero se requiere un criterio de parada. Uno sería cuando encuentre una lista que resuelve el problema, pero en general este no es el caso. Pero si se desconoce la mejor solución, ¿cómo puede el programa determinar si ha encontrado la mejor respuesta, o por lo menos, una de las mejores respuestas? Un primer enfoque sería resolver para una solución fija, que encuentre algún valor predeterminado como mínimo aceptable. Un segundo enfoque es el de correr el programa hasta que caiga su tasa para encontrar nuevas respuestas mejores, o la tasa de mejoramiento de esas respuestas se disminuya drásticamente.

El ejemplo de la mochila se corre hasta que se encuentre una respuesta conocida, en este caso el c romosoma óptimo es: [1, 1, 1, 0, 1, 0, 1, 1], con una aptitud de 35. Desde luego, este problema es l o suficientemente pequeño para resolver con métodos tradicionales, pero sirve para observar cóm o opera el código en forma detallada y darle una buena idea de cómo trabaja el AG.



3.9 Algoritmos genéticos multi-objetivo

Los AGs multi-objetivo, MOGA, son algoritmos que abordan varios objetivos, algunos de los cuales pueden estar en conflicto, aunque se requiere que se optimicen simultáneamente. El conjunto de soluciones óptimas para cada objetivo se conoce como el frente de Pareto. El frente de Pareto entonces comprende soluciones no dominantes, dando a entender que ninguna solución es mejor en todos los objetivos simultáneamente.

Veamos un ejemplo muy concreto, es el diseño de un traje espacial donde se requiere optimizar el diseño que brinde una protección térmica segura, esto es esencial para proteger al astronauta de las temperaturas extremas a la re-entrada a la tierra, considerando también que el traje se debe usar varias veces. Este requerimiento esta en conflicto con la necesidad de que el peso sea mínimo.

Así que para diseñar el traje se debe decidir de la mejor manera como satisfacer los compromisos entre el peso, la durabilidad y el costo. Veamos las funciones para cada uno de los objetivos:

Minimizar el peso W. El material y el grosor del traje, influyen en el peso.

$$F_W = f_1(material, grosor)$$

Maximizar la durabilidad D. La meta es asegurar que el traje puede soportar las temperaturas extremas a la re-entrada.

$$F_D = f_2(material, temperatura)$$

Minimizar el costo C. El costo asociado con los materiales y el proceso de fabricación son cruciales.

$$F_C = f_3(material, proceso)$$

Con base en las tres funciones anteriores, queremos encontrar el mejor posible material y diseño que cumpla con esos objetivos. Hay que buscar materiales ligeros, que puedan soportar excesos de temperatura varias veces y que no sea demasiado costoso.

Se obtienen tres mejores individuos cada uno optimizando una de las funciones anteriores. Con base en esos tres individuos se busaca una solución balanceada de los objetivos; o se discuten los compromisos de cada solución en el contexto del problema que se está abordando.

3.10 Aplicaciones

Optimización de funciones numéricas.

Los AGs se han utilizado mucho en esta tarea, para la solución de ecuaciones especialmente complicadas, como puede ser la búsqueda de raíces en polinomios de 3er. grado en adelante. O en la solución de ecuaciones diferenciales donde se hace necesaria la solución numérica. También en funciones no continuas o funciones con mucho ruido. Por otra parte los costos de los materiales y de producción del traje también se deben tratar de minimizar.

Procesamiento de imágenes

Hay muchas aplicaciones que van desde el arte, generando nuevas expresiones artísticas hasta sistemas de ayuda para la obtención de las caras de sospechosos a partir de los detalles que puede suministrar un testigo. El AG puede generar caras que se pueden ir acercándose a los perfiles descritos, en este caso es el humano el que define la aptitud de los seres cara.

Optimización Combinatoria

Es la tarea de encontrar soluciones a problemas que involucran configuraciones de objetos discretos. A mi modo de ver es uno de los nichos donde mejor se desempeñan los AGs. Son muchos los ejemplos, tales como el problema de la mochila, el problema del agente viajero, programación de tareas en máquinas y muchos otros.

Diseño

En diseño también se mezclan muchos problemas de optimización combinatoria. Por ejemplo, diseño de puentes, diseño de sistemas de distribución de fluidos, y otros.

3.10 Ejercicios y Problemas.

1**.** Maximizar la función $f(x) = x\ sen(10\ \pi\ \text{x}) + 1$, con $x\ \in [0,1]$.

2. Verdadera democracia. Suponga que usted es el jefe de gobierno y está interesado en que pasen los proyectos de su programa político. Sin embargo, en el congreso conformado por 5 partidos, no es fácil su tránsito, por lo que debe repartir el poder, conformado por ministerios y otras agencias del gobierno, con base en la representación de cada partido. Cada entidad estatal tiene un peso de poder, que es el que se debe distribuir. Suponga que hay 50 curules, distribuya aleatoriamente, con una distribución no informe entre los 5 partidos esas curules. Defina una lista de 50 entidades y

asígneles aleatoriamente un peso político de 1 a 100 puntos. Cree una matriz de poder para repartir ese poder, usando AGs.

3. Una empresa proveedora de energía eléctrica dispone de cuatro plantas de generación para satisfacer la demanda diaria de energía eléctrica en Cali, Bogotá, Medellín y Barranquilla. Cada una puede generar 3, 6, 5 y 4 GW al día respectivamente. Las necesidades de Cali, Bogotá, Medellín y Barranquilla son de 4, 3, 5 y 3 GW al día respectivamente. Los costos por el transporte de energía por cada GW entre plantas y ciudades se dan en la siguiente tabla:

	Cali	Bogotá	Medellín	Barranq.
Planta C	1	4	3	6
Planta B	4	1	4	5
Planta M	3	4	1	4
Planta B	6	5	4	1

Los costos del KW-H por generador se dan en la siguiente tabla:

Generador	$KW-H
Planta C	680
Planta B	720
Planta M	660
Planta B	750

Encontrar usando AGs el mejor despacho de energía minimizando los costos de transporte y generación.

4. Genere aleatoriamente una población de 50 matrices de 120 por 180, con números de 0 a 255, preséntelas como una gráfica RGB. La función de aptitud es una imagen cualquiera. Evolucione la población inicial hasta llegar a la imagen.

5. Genere aleatoriamente una población de 50 palabras, que se escuche por el parlante del computador. Tomando como función de aptitud una palabra suya, usando AGs, con base en las palabras generadas aleatoriamente llegue a la palabra que usó como función de aptitud.

6. Tome el algoritmo de la dieta y ahora incluya costos. Ahora encuentre una dieta que trate de satisfacer la dieta pero con un costo mínimo. Este es un ejemplo de AG multi-objetivo con dos funciones objetivo.

Desarrolle tres ejercicios de los 6 propuestos, utilice todas las librerías y herramientas disponibles.

3.9 Bibliografía

Cardona, Daniel, P NOT EQUAL TO NP, Universidad Nacional, 2024, https://arxiv.org/pdf/2401.15079.pdf

Goldberg David, Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, 1989.

Haupt Randy et al, PRACTICAL GENETICALGORITHMS, JOHN WILEY & SONS, INC., 2004

Holland John, Natural and Artificial Systems, an Introductory Analysis with Applications to Biology, 1995. 2nd. Edition, 1995

Mitchell Melanie, An Introduction to Genetic Algorithms, MIT Press, 1999.
Martínez José J., Rojas Sergio, Introducción a la Informática Evolutiva, Nuevo Enfoque para Solucionar Problemas de Ingeniería, Facultad de Ingeniería, Universidad Nacional de Colombia, 2000.

4. PROGRAMACION GENETICA CONCEPTOS BASICOS

José J. Martínez P.
Jmcursosingenieria111@gmail.com
Febrero 26

4.1 Programación Genética

Hablamos de los Algoritmos Genéticos, como algoritmos de búsqueda estocástica, que se basan en los mecanismos de la selección natural, en los que sus cromosomas son listas. La Programación Genética, PG, es un retoño de los Algoritmos Genéticos, en el que sus cromosomas, que sufren la adaptación, son árboles que representan programas. Es decir, son en sí mismos programas de computador.

En Programación Genética, PG, se usan operadores genéticos especializados que generalizan la recombinación sexual y la mutación, para los programas de computador estructurados en forma de árbol que están bajo adaptación. La PG, tiene las siguientes características:

- **Material genético no lineal y estructurado en árbol.**
  La PG opera sobre material genético no lineal, y explícitamente en estructuras de árbol.

- **Material genético de longitud variable.**
  El material genético puede variar de tamaño. Por razones prácticas, se limita el crecimiento de los árboles, aunque se permiten crecimientos a partir de la generación original que se produce aleatoriamente.

- **Material genético ejecutable.**
  La PG es la evolución directa de *programas* de computador. Así, en casi todos los casos el material genético es en cierto sentido ejecutable. Generalmente las estructuras las interpreta algún interpretador, en un lenguaje idéntico o muy parecido a un lenguaje de computación, o a veces en un lenguaje diseñado para el problema a resolver. En casi todos los casos hay un concepto de *ejecución* del material genético, con el objeto de ver el comportamiento de la función deseada, para obtener la aptitud. En otras palabras, el genotipo es equivalente al fenotipo.

- **El Cruce preserva la sintaxis.**
  El cruce preserva la corrección sintáctica del programa que es el material genético, definido para cualquier lenguaje que se haya escogido para su representación.

4.2 Conceptos básicos de PG

El espacio de búsqueda en la PG es el espacio de todos los posibles programas de computador compuestos de funciones y terminales apropiados al dominio del problema. Las funciones pueden ser operaciones aritméticas, operaciones de programación, funciones matemáticas, funciones lógicas, o funciones específicas del dominio.

La PG es un intento de tratar con uno de los aspectos centrales en ciencias de la computación:

> *¿Cómo pueden aprender los computadores a solucionar problemas sin que se les programe explícitamente? En otras palabras, ¿cómo podemos hacer para que los computadores hagan lo que tienen que hacer, sin necesidad de decirles exactamente cómo lo deben hacer?*

Es importante anotar los desarrollos de IA Generativa en la producción de código como de Chat-GPT3, aiXcoder, Mintlify, Copilot, que generan código con un enfoque completamente diferente al de la PG. En sus últimas aplicaciones la PG se utiliza en la producción de lenguajes funcionales. Volveremos luego con este tema.

Para aplicar la PG a un problema determinado, hay cinco pasos preparatorios importantes que se deben definir:

1. El problema
2. El conjunto de terminales,
3. El conjunto de funciones primitivas,
4. La medida de la aptitud,
5. Los parámetros para controlar la corrida, y
6. El método para designar un resultado y el criterio para terminar la ejecución del programa

Paso 1. Debe haber claridad con respecto con respecto al problema que se pretende solucionar por PG, como puede ser el análisis de datos a través de regresiones, clasificación y series de tiempo; entre otros

Paso 2. Los terminales son las hojas de los árboles que corresponden a variables o constantes, son las entradas al programa.

Paso 3. Consiste en identificar el conjunto de funciones $\boldsymbol{F}$ que se va a utilizar, para generar la expresión matemática que trata de satisfacer la muestra dada finita de datos, o las expresiones propias del dominio. A cada función le corresponde una aridad (número de parámetros) y un tipo de datos. Ejemplo, la función seno requiere un solo parámetro, que es un número real. La función suma puede tener dos o más parámetros, los cuales pueden ser enteros o reales. Cada programa de computador (árbol de análisis gramatical) es una composición de funciones del conjunto de funciones F y terminales T.

Cada una de las funciones de F debe aceptar, como argumentos, cualquier valor y tipo de dato, que pueda retornar otra función de F, y cualquier valor que pueda tomar de cualquier terminal T. Esto es, el conjunto de funciones y el conjunto de terminales deben tener la propiedad de clausura.

La PG comienza con una población inicial de programas de computador generados aleatoriamente compuestos de funciones y terminales apropiadas al dominio del problema. La creación de esta población inicial es, en efecto, una búsqueda aleatoria ciega en el espacio de búsqueda del dominio del problema, representado como programas de computador.

Paso 4. Cada programa de computador individual en la población, se mide en términos de qué tan bien se comporta en el ambiente del problema particular, la *medida de aptitud*. La naturaleza de la medida de aptitud varía con el problema.

En muchos problemas la aptitud de un programa se mide ponderando el error al ejecutarlo. El mejor programa de computador tendrá un error cercano a cero. En control óptimo, la aptitud de un programa puede ser el tiempo, combustible o dinero que consume el sistema. En reconocimiento de patrones o clasificación, la aptitud de un programa se mide por una combinación del número de instancias manejadas correctamente y número de instancias manejadas incorrectamente.

En muchos casos, cada programa de la población se corre para varios *casos de aptitud*, de manera que su aptitud se mide como una suma o producto de varias situaciones representativas. Por ejemplo, la aptitud de un programa se puede medir en términos de la suma del valor absoluto de las diferencias entre la salida producida por el programa y la respuesta correcta.

Los programas de la población inicial, generación 0, tienen una aptitud muy pobre. No obstante, algunos individuos serán más aptos que otros. La operación reproducción incluye la selección de un programa de la población actual de programas, con base en su aptitud, permitiéndole si es del caso, sobrevivir copiándolo en la nueva población.

La operación cruce se usa para crear nuevos hijos programas, a partir de dos programas padres seleccionados. Los programas padres, normalmente son de diferentes tamaños y formas. Los programas hijos se componen de subexpresiones, subárboles, subprogramas, de sus padres. Así, los programas hijos normalmente son de diferentes tamaños y formas que las de sus padres.

Intuitivamente, si dos programas tienen alguna efectividad en solucionar un problema, entonces alguna de sus partes probablemente tenga algún mérito. La recombinación de partes escogidas aleatoriamente, de programas con alguna efectividad, a veces produce nuevos programas de que son aún más aptos para solucionar un problema determinado, que cualquiera de sus padres.

Paso 5. Después de efectuar las operaciones genéticas sobre la población actual, la población de hijos reemplaza la generación anterior, a cada individuo de la nueva población se le mide su aptitud, y el proceso se repite por muchas generaciones. En cada etapa de este proceso, el estado del proceso consiste de la población actual de individuos. La fuerza que controla este proceso consiste solo en la aptitud observada de los individuos en su lucha con el ambiente problema. Los parámetros que controlan la ejecución de PG son: el tamaño de la población; el número de generaciones; la probabilidad de aplicación del operador de mutación; y la estrategia de selección de los individuos para la nueva generación.

Paso 6. El mejor individuo que aparece en cualquier generación de una corrida, es decir el mejor hasta ahora, normalmente se designa como el resultado producido por la corrida de PG.

Otra característica de la PG es la ausencia de pre-procesamiento de las entradas y el pos-procesamiento de salidas, ya que el genotipo y el fenotipo son lo mismo. Las entradas, resultados intermedios, y salidas se expresan normalmente directamente en términos de la terminología del dominio del problema. El pos-procesamiento de la salida, si lo hay, se hace con una interface de salida.

4.3 Solución de problemas por PG En resumen, la PG incuba programas de computador para la solución de problemas, ejecutando los siguientes pasos:

1. Generar una población inicial de composiciones aleatorias de funciones y terminales del problema (es decir, programas).

2. Ejecutar iterativamente los siguientes pasos hasta que se satisfaga el criterio de terminación:

a. Ejecutar cada programa de la población y asignarle un valor de aptitud, de acuerdo con su comportamiento frente al problema.

b. Crear una nueva población de programas aplicando las siguientes dos operaciones primarias, a los programas escogidos, con una probabilidad basada en la aptitud.

i. Reproducir un programa existente copiándolo en la nueva población.

ii. Crear dos programas a partir de dos programas existentes, recombinando genéticamente partes escogidas de los dos programas en forma aleatoria, usando la operación cruce aplicada a un punto de cruce, escogido aleatoriamente dentro de cada programa.

iii. Crear un programa a partir de otro seleccionado aleatoriamente, cambiando aleatoriamente un gen (función o terminal). Es posible que se requiera un procedimiento de reparación para que se satisfaga la condición de clausura.

3. El programa identificado con la mayor aptitud (el mejor hasta la última generación), se designa como el resultado de la corrida de PG. Este resultado puede representar una solución (o una solución aproximada) al problema.

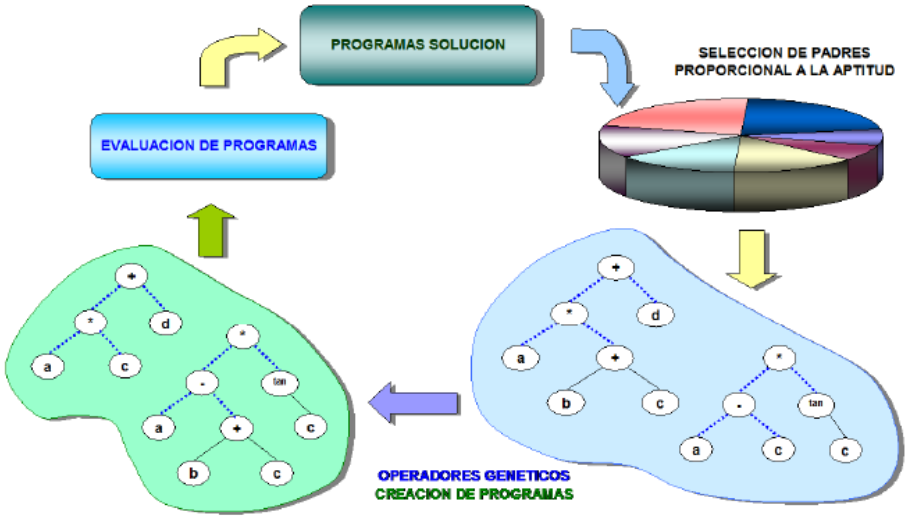


***Figura 4.1**. Estructura general de Programación Genética*

4.3.1 Ejemplo del operador cruce

El cruce genético (recombinación sexual) opera sobre dos programas padres y produce dos nuevos programas hijos consistentes de partes de cada padre.

Consideremos los siguientes programas:

$(a + b) * (c \uparrow e) + f$ y $(c * f) + ((a + b) \uparrow d)$

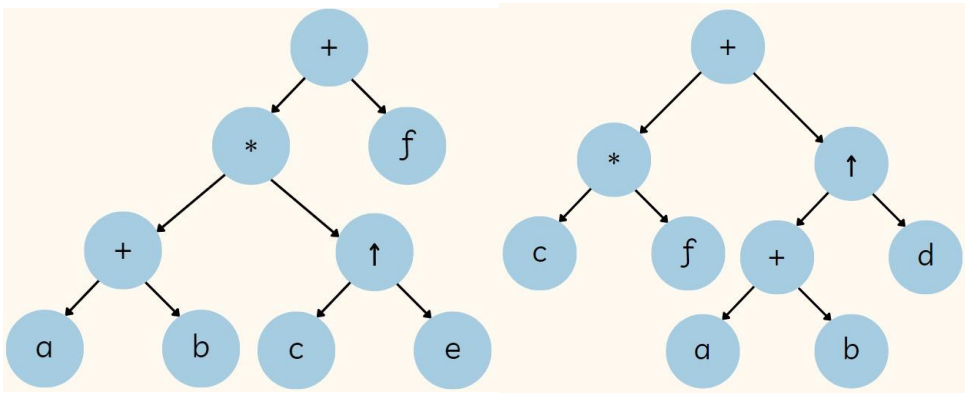


Figura 4.2. *Programas de computador padres*

En la figura 4.2 estos dos programas se presentan como árboles rotulados con ramas ordenadas. Los nodos, del árbol corresponden a funciones (es decir, operaciones) y las hojas, corresponden a terminales (es decir, datos de entrada). La operación cruce crea nuevos hijos, intercambiando subárboles (es decir, subrutinas, sub-procedimientos) entre los dos padres, figura 4.3.

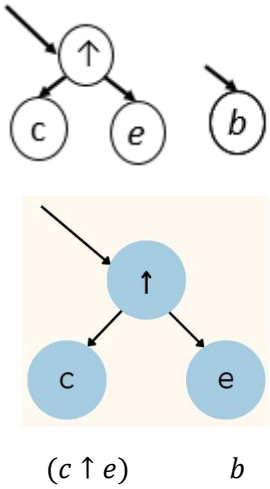


Figura 4.3. *Dos fragmentos de cruce.*

Suponga que los nodos de ambos árboles se numeran en preorden (raíz, subárbol izquierdo, subárbol derecho). Suponga también, que se seleccionó aleatoriamente el nodo número 6, entre los 9 nodos del primer árbol, como punto de cruce para el primer padre y que se seleccionó aleatoriamente el nodo número 5, entre los 9 nodos del segundo árbol, como punto de cruce del segundo padre. Por lo tanto, los puntos de cruce son el ↑ para el primer padre y el operando b para el segundo padre.

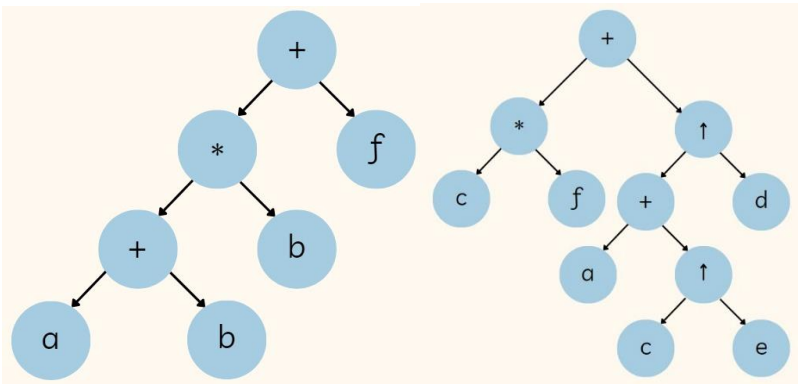


Figura 4.4*. Dos descendientes.*

Los hijos resultantes del cruce se muestran en la figura 4.4, $(a + b) * b + f$ y $(c * f) + (a + (c \uparrow e) \uparrow d)$

De esta manera, el cruce crea nuevos programas usando partes de los programas padres existentes. Debido a que se intercambian subárboles completos, la operación cruce siempre produce programas sintáctica y semánticamente válidos. Debido a que los programas padres se han seleccionado para la operación cruce con una probabilidad basada en su aptitud, el cruce explora regiones del espacio de búsqueda con programas que contienen partes de programas promisorios.

Otro aspecto importante de observar es que mientras la profundidad de los árboles padres es 4, la profundidad del primer hijo es 4 y la profundidad del segundo hijo es 5. Esto lleva a que al avanzar el proceso evolutivo aumenta la profundidad de los árboles, lo que puede llevar a la interrupción del programa por el manejo de sus pilas.

4.4 Estructuras de datos y operadores en PG

4.4.1 Mecanismo de codificación

En toda actividad que necesite la ejecución de una secuencia especial de pasos o tareas se tienen algoritmos y programas de computador. La PG busca representaciones simples de tareas sencillas "que evolucionen" a formas más complejas, que permitan resolver el problema de la mejor manera. La forma más común de representar estos programas es mediante funciones y parámetros. En teoría de compiladores y lenguajes formales, los algoritmos y las funciones, se pueden visualizar y se representan por medio de árboles.

Un árbol es una estructura de datos, que se define como "un conjunto finito de nodos T tales que: a) Hay un nodo raíz. B) Los otros nodos se particionan en M conjuntos $T_1, T_2, \ldots, T_M$. Los conjuntos $T_1, T_2, \ldots, T_M$ son árboles y se llaman subárboles de T. Cuando un árbol representa un programa, sus nodos internos representan funciones y las hojas representan terminales o datos de entrada.

4.5. Representación y recorrido de árboles binarios

El manejo de la estructura árbol es básico en PG, y por eso lo vamos a estudiar con algún detalle. Los árboles se pueden representar en memoria estática a través de estructuras tipo matriz o a través

de estructuras dinámicas. Aquí se trabajará el manejo dinámico; así comenzamos definiendo el nodo de un árbol:

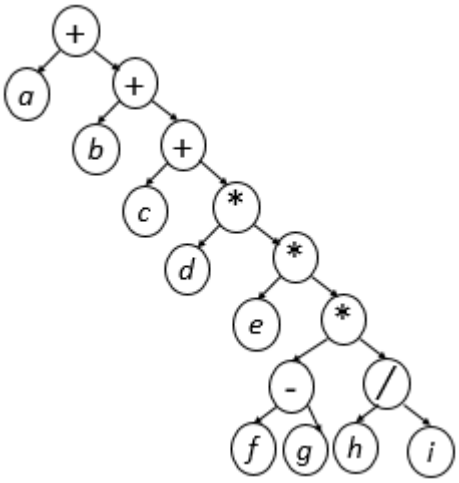


***Figura 4.7**. Árbol binario equivalente al árbol no binario de la figura 8.5.*

Enlace izquierdo	Informa-ción	Enlace derecho

***Figura 4.8**. Nodo de un árbol binario*

Una operación muy importante es el recorrido de los árboles binarios. Recorrer un árbol significa visitar los nodos del árbol en forma sistemática, de esta manera se garantiza que se visitan todos los nodos una sola vez.

Hay tres métodos, recursivos, para hacer el recorrido de un árbol:

Preorden: Raíz, Subárbol Izquierdo, Subárbol Derecho.
Inorden: Subárbol Izquierdo, Raíz, Subárbol Derecho.
Postorden: Subárbol Izquierdo, Subárbol Derecho, Raíz.

Ejemplo. Se hacen los recorridos con respecto al árbol de la figura 4.9.

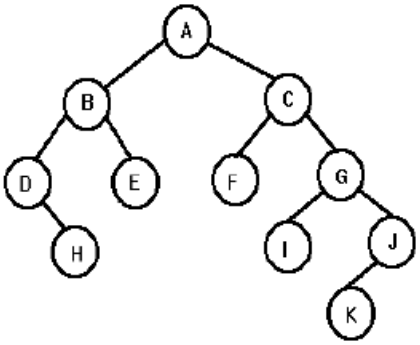


***Figura 4.9.** Árbol binario de ejemplo para hacer los recorridos.*

PREORDEN: ABDHECFGIJK.
INORDEN: DHBEAFCIGKJ.

POSORDEN: HDEBFIKJGCA.
Veamos el recorrido de un árbol en preorden. Se entra al árbol por el apuntador T que apunta a la raíz.

Un algoritmo muy útil en PG, es **Cuenta_Nodos,** que realiza el conteo de los nodos de un árbol, y es básicamente el algoritmo anterior, solo que en lugar de visitar los nodos los va contando.

4.6 Generación de árboles

A continuación, se presenta una forma de generación de árboles programa, teniendo como base el conjunto de funciones F y el conjunto de terminales T. Se supone un arreglo de Funciones, donde están las funciones y un arreglo de Terminales, donde están los terminales.

El primer algoritmo genera un árbol binario compuesto de raíz y dos hojas. Lo llamamos Genera_Subárbol. La raíz es una función escogida aleatoriamente del conjunto de Funciones, y sus dos hojas corresponden a dos terminales seleccionados aleatoriamente. Se supone que hay ***i*** funciones y ***j*** parámetros.

Un árbol se genera llamando iterativamente **Genera_Subárbol.** El subárbol T1 creado en **Genera_Subárbol,** reemplaza una hoja o terminal del árbol que se está generando. Este terminal se escoge aleatoriamente. El número de subárboles también puede ser aleatorio, sin embargo, no es necesario que el árbol tenga tanta profundidad. Se da como parámetro el número de nodos que se quiere, ***n***.

4.7 Cruce de árboles

Es el operador más importante en PG es el cruce. En cada generación dos individuos de los más aptos combinan su información genética, formando dos nuevos individuos. Por ejemplo, si tenemos las dos frases “Me gusta dormir tarde los domingos” y “Mi padre habla acerca del clima con mi novia” y las cruzamos (con un punto de cruce, como por ejemplo la cuarta y sexta palabras), se obtienen dos nuevas frases, con significados diferentes. Lo importante aquí es notar cómo el cruce induce a la variedad en una población evolutiva, permitiendo llegar a soluciones.

Una vez se tiene una población de árboles programa, se ejecutan y se mide la aptitud de cada uno, con respecto al problema que se quiere solucionar. Con base en esta medida de aptitud, se seleccionan los árboles programa padres para la siguiente generación. Hay varios métodos para hacerlo, que se pueden encontrar en la bibliografía de AG, por lo que aquí no nos vamos a ocupar de ellos.

Inicialmente, se escoge una pareja de individuos para cruzar, y aleatoriamente se escoge un punto en cada uno de ellos un punto de cruce. Se hace un traslado completo de subárboles de un árbol al otro y viceversa. Tomando la numeración de los nodos en preorden, se intercambian los subárboles. En la figura 4.10 vemos un ejemplo de cruce con dos programas muy sencillos (¿cuál sería el listado de código de estos programas?). Los nodos están enumerados en preorden.

Los pasos para realizar el cruce de dos árboles programas en PG son:

1. Seleccionar árboles a cruzar
2. Encontrar el número de nodos de cada árbol
3. Seleccionar el punto de cruce para cada árbol
4. Realizar el intercambio de subárboles por el punto de cruce escogido en cada árbol.

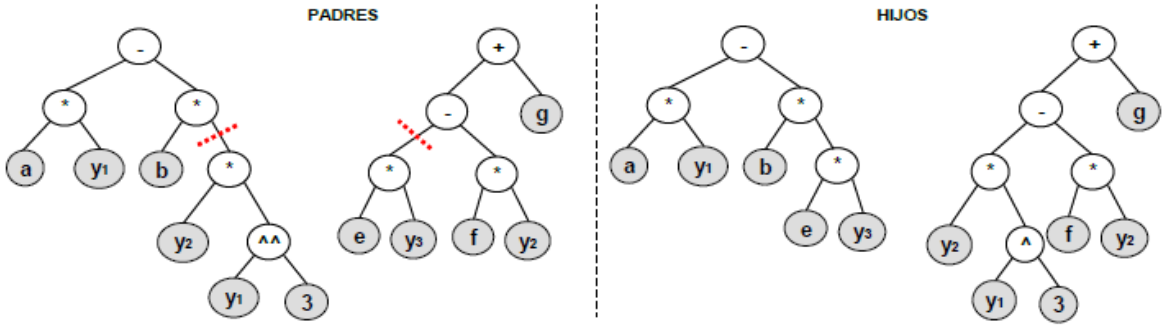


Figura 4.10. *Otro ejemplo del operador de cruce*

1. Seleccionar árboles a cruzar
Se escogen de la población de árboles seleccionados por su aptitud, aleatoriamente dos árboles, **T1** y **T2**.

2. Encontrar el número de nodos de cada árbol
Para cada árbol se cuenta el número de nodos que tiene con el algoritmo **Cuenta_Nodos (T,n).** Se corre para **T1** y **T2**, obteniendo respectivamente el número de nodos **n1** y **n2**.

3. Seleccionar el punto de cruce para cada árbol
Con base en n1, se escoge un punto de cruce **c1** para **T1** y con base en **n2**, se escoge un punto de cruce **c2** para **T2**. Se recorre cada árbol hasta el punto de cruce, encontrando el apuntador a ese nodo. Esto se hace con el algoritmo **Punto_Cruce (T, c, q)** que obtiene el apuntador **q** que direcciona el nodo **c** y la variable**. Ind_Enlace** nos indica porque subárbol de **q.**

4. Realizar el intercambio de subárboles
Una vez se tiene el apuntador de cada punto de cruce en su respectivo árbol, se intercambian los apuntadores, realizándose de esta manera el cruce.

Los algoritmos de mutación son equivalentes a los que se utilizan en el cruce (búsqueda del nodo) con la diferencia de que se debe cambiar el nodo objetivo por otro. Hay que tener la precaución de cambiar una función de ***i*** parámetros por una equivalente con los mismos parámetros y tipo, o una constante o variable por otra constante o variable.

4.8 Ejecución del programa árbol
Una vez se tiene la nueva población de árboles se deben correr para conocer cuál es su aptitud con respecto a los criterios que se han fijado. Para correr un programa, primero se convierte a notación polaca y luego propiamente se ejecuta la expresión polaca. La expresión polaca se obtiene recorriendo el árbol en posorden.

Ejemplo, el recorrido en posorden del árbol de la figura 4.7 es el siguiente:

$abcdefg - hi/*** + + +$

Una vez se tiene la expresión en posorden se realiza la evaluación. Para esto se supone que la expresión está en el vector X, con n elementos.

Los algoritmos se presentan en el Notebook de Jupyter.

4.9 Aplicaciones

Síntesis general de programas

Es un área cada vez más importante de aplicación de la PG. El CBGP (Code Building Genetic Programming) permite hacer síntesis general de programas que soporta la evolución de programas que pueden usar tipos de datos arbitrarios y polimorfismo. Entre estos se encuentran varias herramientas CAD como son: Clojure, es un lenguaje dinámico de propósito general, dialecto de LISP; OpenSCAD, un compilador 3D; QCad, que es una CAD para 2D; FreeCAD, es un modelador paramétrico 3D; e InkScape, que es un editor de gráficos vectoriales.

Programación de FPGAs

El hardware evolutivo es un concepto relativamente nuevo y moderno en el diseño de circuitos, que se basa en estructuras reconfigurables. Los sistemas evolutivos pueden cambiar dinámicamente su función y sus parámetros. En este sentido los FPGAs son muy útiles.

Regresión Simbólica

La regresión simbólica consiste en obtener una función algebraica que ajuste en gran medida un conjunto de datos experimentales. Similar a la muy conocida regresión por mínimos cuadrados, la diferencia es que en la regresión simbólica no se conoce el formato de la función a ajustar, por lo tanto, se debe obtener la función y los mejores coeficientes que ajusten los datos experimentales.

De los datos experimentales se pueden reconocer las variables dependientes e independientes del problema, en el caso de la figura 4.11, existe una variable dependiente y y dos variables independientes x_1, x_2; entonces el conjunto de terminales del problema PG serán las variables independientes y los coeficientes de la función a ajustar. El conjunto de funciones se puede limitar a las funciones elementales: $\{+,\ *,\ /, -\}$. Nótese que el árbol como programa se ejecuta cuando se evalúa el modelo algebraico con los terminales como entradas.

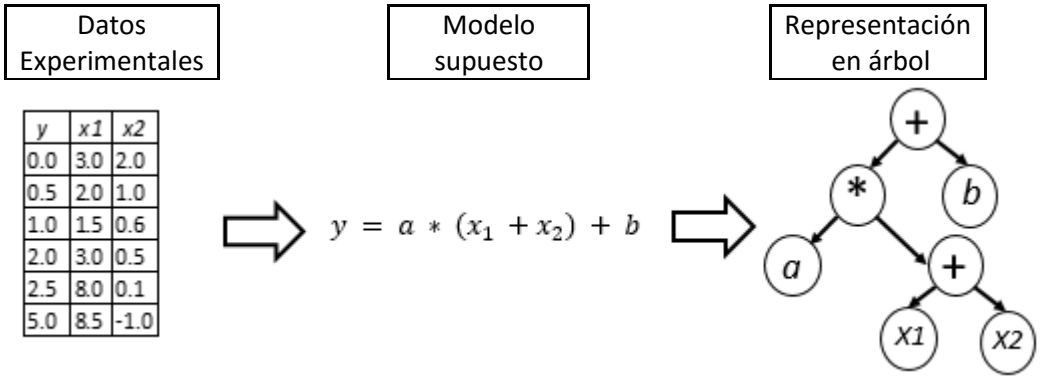


Figura 4.11. *Representación en árbol de un modelo supuesto, para un conjunto de datos experimentales.*

La función de aptitud es entonces un indicativo que permite premiar a los individuos, árboles, que se ajustan mejor a los datos experimentales. Para encontrar una función de aptitud útil se puede utilizar el error absoluto, la diferencia de valores evaluados con el modelo supuesto y los datos experimentales. El error absoluto es menor para los árboles más aptos, de manera que el error es inversamente proporcional a la aptitud, por lo que definimos la función de aptitud f_{apt},

$$f_{apt} = \frac{1}{\sum_j \left| y_j - yc_{i,j} \right|}$$

Donde los y_j son los valores de la tabla, en este caso $j = 1, \dots, 6$. Y los $yc_{i,j}$ son los valores calculados por todos los arboles i de la población, para la posición j de la tabla. Así que f_{apt} indica que la aptitud del $i - ésimo$ árbol es igual al inverso de la suma de errores absolutos (donde es el valor del $j - ésimo$ dato experimental evaluado en las variables independientes; mientras que, es el valor evaluado en el $i - ésimo$ árbol; por lo tanto, si el error es pequeño la aptitud será muy grande. De manera que, si el error es cero, el valor de la función de aptitud tenderá a infinito; esto se evita en la práctica colocando en el denominador un número relativamente pequeño para fijar un límite superior:

$$f_{apt} = \frac{1}{(\,0.1 + \sum_j \left| y_j - yc_{i,j} \right|)}$$

Entonces por la ecuación anterior tendrá como máxima aptitud f_{apt}, el valor de 10, para un error cero.

Supongamos que se ha corrido un ejemplo para el mismo programa, árbol, de la figura 8.11, para valores obtenidos de $a = 1.5$ y $b = -1.0$, se han encontrado los valores yc respectivos. La diferencia en valor absoluto de $\left| y_j - yc_{i,j} \right|$ es el error. La f_{apt} se encuentra haciendo primero la sumatoria del error para cada valor de la tabla, 6.5, luego se suma el 0.1 y se saca el valor inverso para obtener $f_{apt} = 0.1515$

yc	*x1*	*x2*	Error
1.5	3.0	2.0	1.5
0.5	2.0	1.0	3E-11
0.05	1.5	0.6	0.95
0.75	3.0	0.5	1.25
3.05	8.0	0.1	0.55
2.75	8.5	-1.0	2.25
Error total			6.5
Aptitud 0.1515			

Figura 4.11. *Representación en un árbol del modelo supuesto, para un conjunto de datos experimentales.*

Los parámetros de control para este ejemplo de PG se fijaron con los valores recomendados [7], probabilidad de cruce = 0.7; probabilidad de mutación 0.01; tamaño de la población= 100, generaciones = 500, Criterio de selección = elitismo [4]. Por lo tanto, el sistema de PG se detiene cuando se cumple el número de generaciones y la solución será el árbol con mejor aptitud.

En el transcurso de las generaciones y gracias al operador de cruce los árboles empiezan a crecer especialmente en profundidad, generando árboles muy profundos que aportan muy poco al incremento de la aptitud. Así que es conveniente fijar un tamaño máximo para el tamaño de los árboles de la población, o incluso un cruce que haga esta limitación.

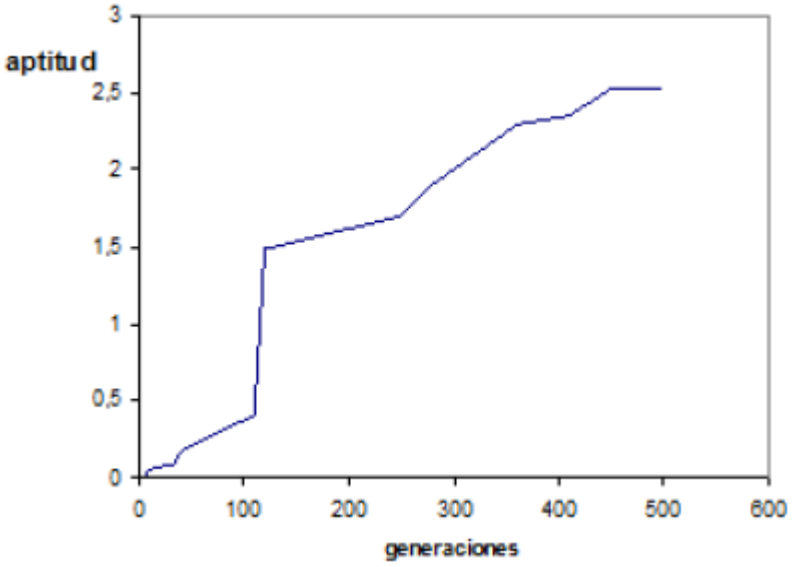


***Figura 4.12**. Historia de la aptitud para una regresión simbólica*

En la figura 4.12, se puede observar cómo varía la aptitud, del mejor individuo en cada generación, al final se obtiene una aptitud de 2.53456 con la siguiente expresión como solución:

$$y = a * x_1 - b * x_2 * x_2 + c * (d * x_1 + e * x_2)/(x_1 - x_2 + f)$$

4.10. Múltiples expresiones en un solo cromosoma

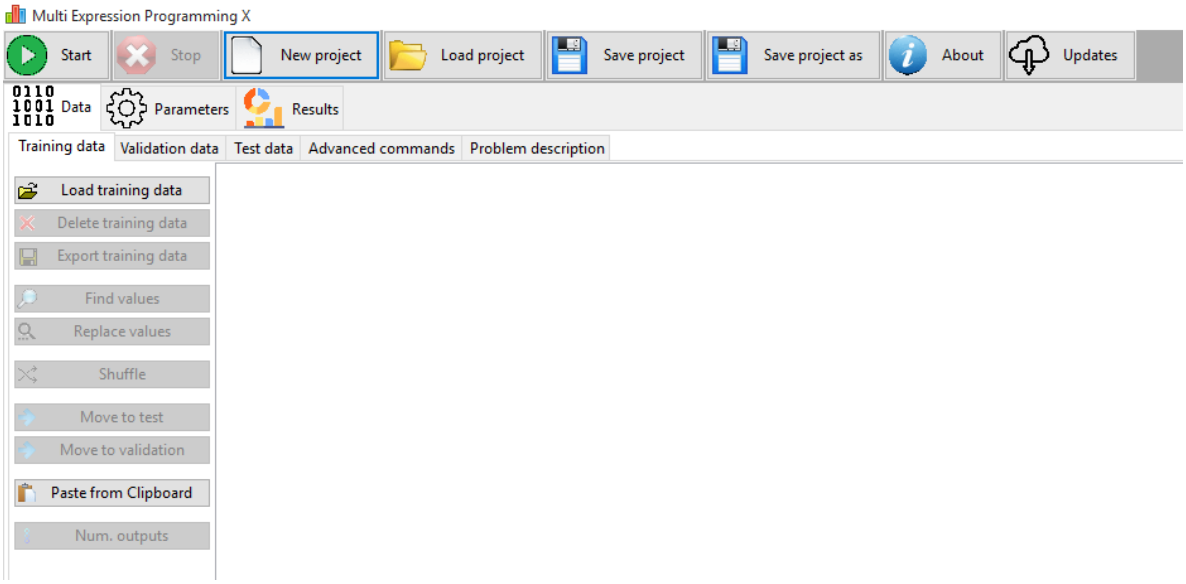
Hay una serie de técnicas de PG, entre las que está la de tener múltiples expresiones en un solo cromosoma. La idea es un cambio en la estructura de los cromosomas en el sentido de tener varias expresiones en el cromosoma, MEP. Lo que significa que podemos hacer una exploración mucho más profunda del espacio de soluciones programa. Su objetico como toda la PG es la generación de programas.

Representación

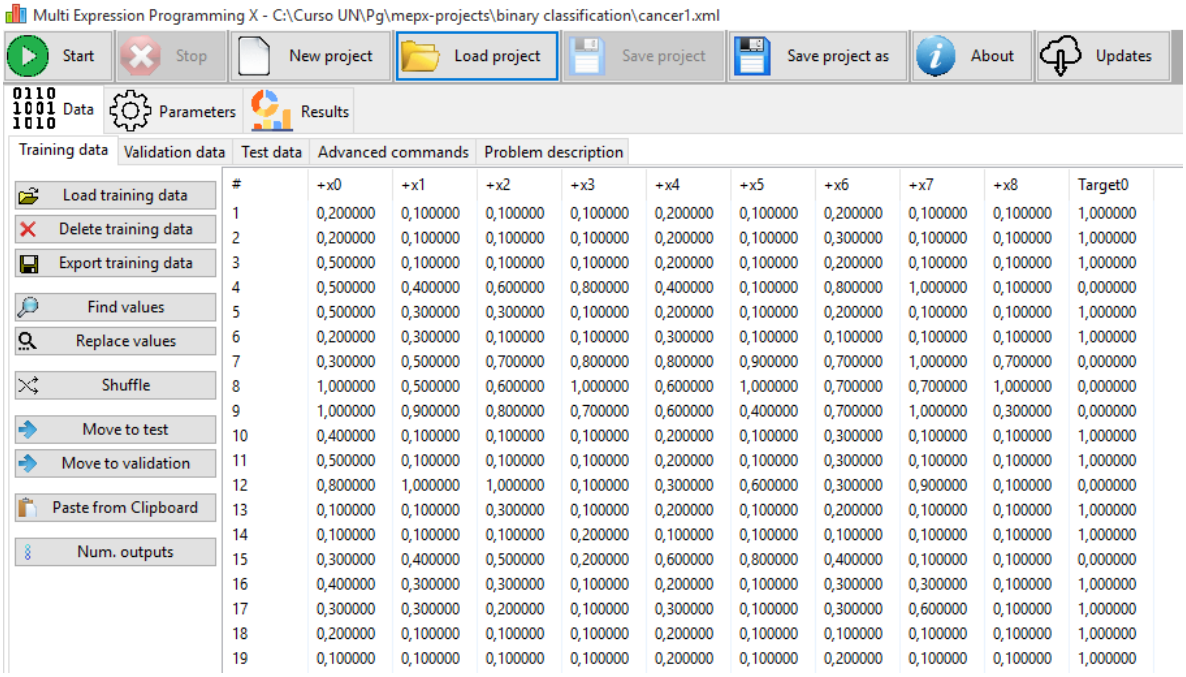
La representación se hace con base en listas que son representaciones de árboles, inspirada en el código three-address, que es un código intermedio que usan los compiladores optimizados para ayudar en la implementación de transformaciones para mejorar el código. Así que cada cromosoma tiene un número constante de genes, donde cada gen corresponde a una instrucción o a una variable, una terminal. Los genes que codifican una función incluyen un apuntador a los argumentos de la función.

MEPX

Un paquete muy interesante y poderoso es el MEPX, con el cual se pueden trabajar una serie de aplicaciones y que se puede descargar de https://www.mepx.org/. Incluye una serie de ejemplos con los que se puede experimentar en clasificación y regresión. La siguiente es la entrada:

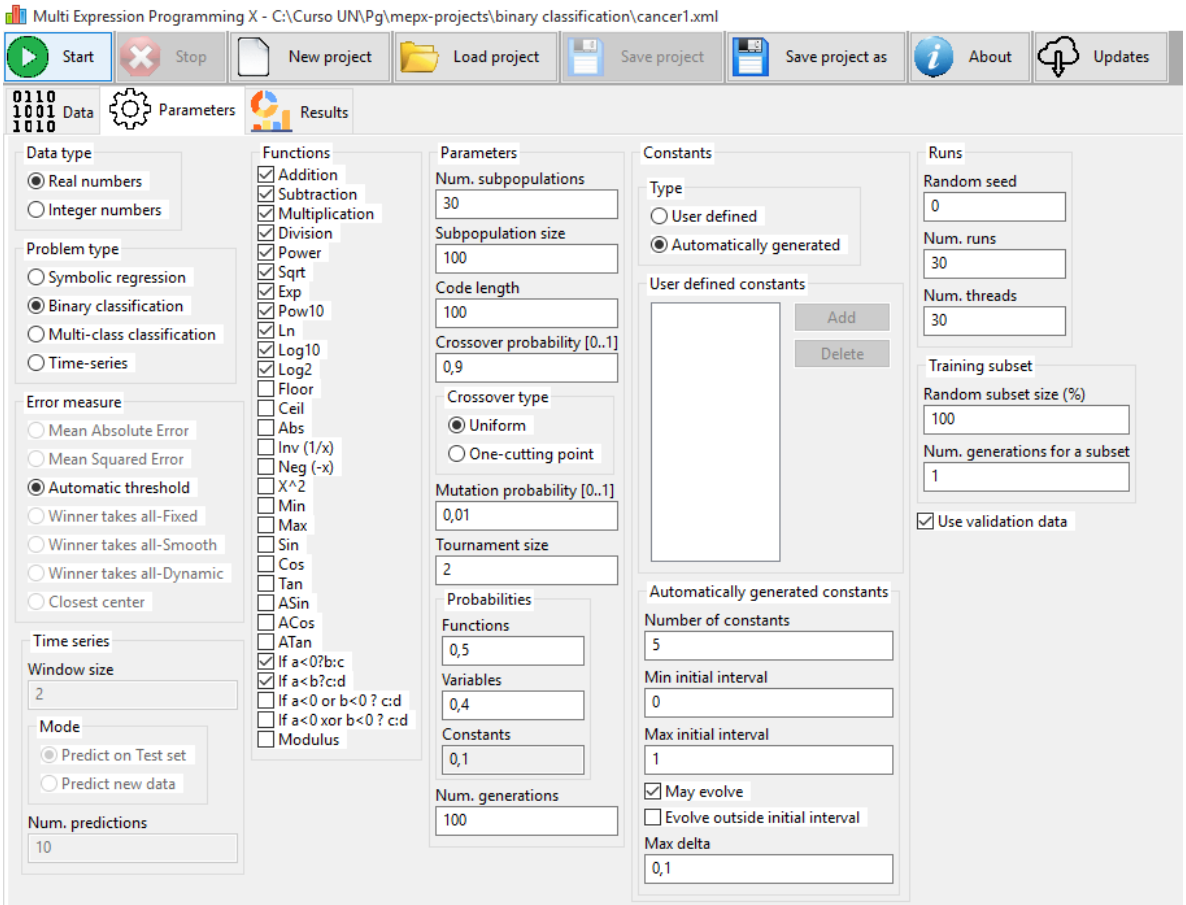


Luego se escoge uno de los ejemplos, por ejemplo "Cáncer 1" que incluye los datos relacionados con las variables, “features”, utilizados para determinar si una persona tiene o no cáncer.



#	+x0	+x1	+x2	+x3	+x4	+x5	+x6	+x7	+x8	Target0
1	0,200000	0,100000	0,100000	0,100000	0,200000	0,100000	0,200000	0,100000	0,100000	1,000000
2	0,200000	0,100000	0,100000	0,100000	0,200000	0,100000	0,300000	0,100000	0,100000	1,000000
3	0,500000	0,100000	0,100000	0,100000	0,200000	0,100000	0,200000	0,100000	0,100000	1,000000
4	0,500000	0,400000	0,600000	0,800000	0,400000	0,100000	0,800000	1,000000	0,100000	0,000000
5	0,500000	0,300000	0,300000	0,100000	0,200000	0,100000	0,200000	0,100000	0,100000	1,000000
6	0,200000	0,300000	0,100000	0,100000	0,300000	0,100000	0,100000	0,100000	0,100000	1,000000
7	0,300000	0,500000	0,700000	0,800000	0,800000	0,900000	0,700000	1,000000	0,700000	0,000000
8	1,000000	0,500000	0,600000	1,000000	0,600000	1,000000	0,700000	0,700000	1,000000	0,000000
9	1,000000	0,900000	0,800000	0,700000	0,600000	0,400000	0,700000	1,000000	0,300000	0,000000
10	0,400000	0,100000	0,100000	0,100000	0,200000	0,100000	0,300000	0,100000	0,100000	1,000000
11	0,500000	0,100000	0,100000	0,100000	0,200000	0,100000	0,300000	0,100000	0,100000	1,000000
12	0,800000	1,000000	1,000000	0,100000	0,300000	0,600000	0,300000	0,900000	0,100000	0,000000
13	0,100000	0,100000	0,300000	0,100000	0,200000	0,100000	0,200000	0,100000	0,100000	1,000000
14	0,100000	0,100000	0,100000	0,200000	0,100000	0,100000	0,100000	0,100000	0,100000	1,000000
15	0,300000	0,400000	0,500000	0,200000	0,600000	0,800000	0,400000	0,100000	0,100000	0,000000
16	0,400000	0,300000	0,300000	0,100000	0,200000	0,100000	0,300000	0,300000	0,100000	1,000000
17	0,300000	0,300000	0,200000	0,100000	0,300000	0,100000	0,300000	0,600000	0,100000	1,000000
18	0,200000	0,100000	0,100000	0,100000	0,200000	0,100000	0,100000	0,100000	0,100000	1,000000
19	0,100000	0,100000	0,100000	0,100000	0,200000	0,100000	0,200000	0,100000	0,100000	1,000000

Presenta la operaciones que se van a utilizar seleccionadas de un conjunto de todas las posibles operaciones que se pueden utilizar.



Se inicia el entrenamiento del modelo utilizando los parámetros previamente definidos. El m odelo se ajusta iterativamente a los datos de entrenamiento, y se calcula un error con respec to a la validación para evaluar su rendimiento.

MEPX proporciona métricas de evaluación del rendimiento del modelo, como el error prome dio en comparación con los datos de validación. Esto nos permite determinar qué tan bien se desempeña el modelo en la tarea de clasificar la presencia o ausencia de cáncer.

Después del entrenamiento exitoso, MEPX genera una salida que proporciona detalles sobre el número de generaciones y corridas realizadas. También genera automáticamente un códig o en Python (cáncer.py en este caso) que se puede utilizar para aplicar el modelo entrenado para llevarlo a otros sistemas o plataformas.

4.11 Relación entre PG e IA Generativa para la generación automática de programas.

La generación automática de programas ha sido un campo de estudio e investigación en IA. El enfoque original era utilizar la PG. Sin embargo, con la aparición de la IA Generativa, con modelos de lenguaje especializados, se han superado una serie de barrera que todavía se mantenían. Veamos la siguiente tabla comparativa.

Aspecto	Programación Genética (GP)	Modelos Generativos de IA (LLMs para código)

Inspiración	Evolución biológica	Aprendizaje estadístico profundo, modelos de lenguaje
Generación de código	Evolución de árboles	Predicción de código basado en contexto
Representación interna	Árboles sintácticos o estructuras de instrucciones	Texto plano (tokens de código fuente)
Mecanismo de mejora	Selección natural basada en aptitud	Aprendizaje supervisado
Exploración del espacio	Estocástica y evolutiva	Probabilística, basada en patrones del corpus de entrenamiento
Objetivo	Encontrar soluciones que optimicen una función objetivo	Generar código correcto y relevante según el prompt
Control del proceso	Alto: se define la función de aptitud y operadores genéticos	Bajo: se define el prompt, pero el modelo decide la solución
Personalización	Alta: se puede diseñar la aptitud específica para el problema	Moderada: se puede ajustar vía prompt o fine-tuning
Creatividad / Sorpresa	Alta: puede generar soluciones no convencionales	Alta: puede combinar patrones en formas nuevas
Limitaciones	Requiere muchas evaluaciones y puede ser lento	Puede "alucinar" o generar errores sutiles

Similitudes y Diferencias

Similitudes
En ambos enfoques se busca generar automáticamente programas de computador, código, sin programación manual directa. También se requiere que los programas que generan sean validados por humanos para determinar su calidad y su posible utilización. Los enfoques de ambas permiten generar soluciones creativas no triviales. Mientras en la PG, la generación de programas

Diferencias
La PG se basa en un proceso evolutivo lento, mientras que con los modelos generativos, la predicción es inmediata. En la PG, se tiene una gran población de posibles soluciones que se trabajan en paralelo, mientras que en el enfoque de IA generativa, se genera de a una solución. Mientras que en la PG la generación de programas es altamente controlable e interpretable, en la IA Generativa es opaca, aunque muy rápida.

4.12 Aplicaciones

- Planeación de trayectorias
- Diseño de circuitos lógicos
- Estrategia de juegos
- Programación de máquinas
- Machine Learning
- Programación de computadores
- Reingeniería
- Control óptimo https://www.youtube.com/watch?v=K2Hl7m2Ty_4

4.12. Ejercicios

De los ejercicios 1 y 2, escoja 1 para realizar; y hacer 3 o 4.

1. Descargue MEPX, https://www.mepx.org/, estúdielo y corra uno de los ejemplos que trae.
2. Suponga que desea utilizar Programación Genética para encontrar el diseño de un circuito lógico, tome como, ejemplo el codificador de 7 segmentos. Describa el conjunto de terminales, el conjunto de funciones y la función de aptitud. Use una librería de Python.
3. Suponga que tiene un robot que le entrega galletas al grupo de ingenieros de diseño de robots. Programe por PG el recorrido del robot, teniendo en cuenta que cada vez que un ingeniero recibe una galleta gana puntos. Los ingenieros están distribuidos en una sala cuadrada. Defina, conjunto de terminales, conjunto de funciones y función de aptitud.
4. Vea el video, https://www.youtube.com/watch?v=6KNuJn6dVy4. Analícelo y haga un ejemplo de control aplicando PG.

4.11 Bibliografía

[1] Gen M y Cheng R, *Genetic Algorithms and engineering design*. New York: John Wiley & Sons, 1997.

[3] Koza J, *Genetic Programming, on the programming of computers by means of natural selection*. The MIT Press, 1992

[4] Langdon W.B y Poli R, *Foundations of Genetic Programming* . Berlin: Springer-Verlag, 2002.

[5] Martínez J. Estructuras de Información. Bogotá, Colombia: Universidad Nacional de Colombia. Primera Edición. 2000.

[6] Martínez J y Rojas S. *Introducción a la informática Evolutiva.* Bogotá, Colombia: Universidad Nacional de Colombia. Primera Edición. 1999.

[7] McKay B., Willis M y Barton G, "Steady-state modelling of chemical process systems using genetic programming," *Comp & Chem Eng.*, vol 21, no 9, pp. 981-996, 1997.

[8] Michalewicz Z, *Genetic Algorithms+Data Structures=Evolution Programs* . Berlin: Springer-Verlag, 1994.

[9] Poli R y Langdon W.B. "Schema theory for genetic programming with one-point crossover and point mutation". *Evolutionary Computation*, 6(3): 231-252, 1998.

[10] Torres Juan Esteban, Martínez José Jesús, *Departamento de Ingeniería de Sistemas e Industrial, Universidad Nacional de Colombia, Bogotá,* 2004.

[11] Su Nguyen1, Yi Mei1, Mengjie Zhang1, Genetic programming for production scheduling: a survey with a unified framework, This article is published with open access at Springerlink.com, 2017.

5. INTRODUCCION A LAS REDES NEURONALES ARTIFICIALES

José J. Martínez P.

jmcursosingenieria111@gmail.com
Febrero 2026

5.1 Introducción

Una de las fuentes de datos más importantes de los seres vivos son los datos provenientes del ambiente. Los filtran nuestros ojos, nuestros oídos y también otros sentidos. La principal fuente de supervivencia de las especies proviene de estos datos que internamente analiza cada animal en su cerebro, buscando patrones para determinar si hay presas, calculando el esfuerzo para atraparlas, o si son presas para eludir los predadores. De esa interacción permanente con el ambiente realmente se hace Machine Learning, pues son muchos los datos que llegan permanentemente y se van almacenando, a partir de los cuales se logra el conocimiento. En la figura 5.1, el pájaro observa la mariposa y su modelo mental reconoce el tipo de mariposa, para tomar una decisión.

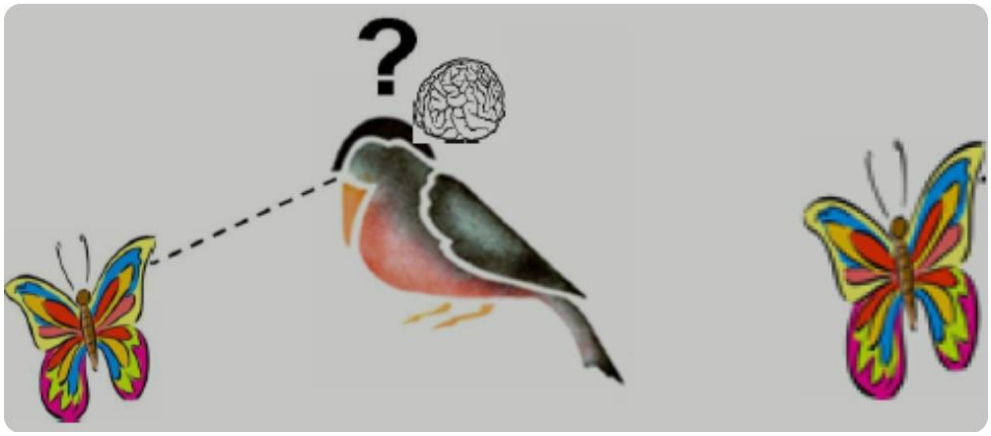


Figura 5.1. Modelo mental para predecir.

El modelo mental elimina detalles no pertinentes y enfatiza en los detalles de interés. En este caso el pájaro, filtra de la cantidad de datos que ofrece el ambiente y con base en su visión y otros sentidos, encuentra patrones y actúa en consecuencia. Cuando el agente no detecta apropiadamente el patrón, pierde esfuerzos y debe realizar cambios en su estructura mental para adaptar el modelo que le permita una mejor anticipación de las consecuencias, cuando se encuentre nuevamente con ese patrón o uno similar. Los agentes que fallan pierden recursos que generalmente favorecen a los exitosos.

Ese cambio en su modelo mental es lo que llamamos aprendizaje a partir de la experiencia. Si se examinan problemas, cuya descripción algorítmica es muy compleja, como identificar en fracciones de segundo una persona entre una multitud, sus soluciones tienen una característica común: la experiencia. Los animales superiores, como el hombre, son capaces de resolver estas situaciones acudiendo a la experiencia acumulada.

Así, de manera muy simplista, es razonable pensar que una forma de solucionar este tipo de problemas es construir sistemas que sean capaces de simular esta característica. La observación anterior fue considerada por los científicos de ciencias de la computación en lo que se denomina Neurocomputación cuyo objeto de estudio son las Redes Neuronales Artificiales, RNAs. Las RNAs no son más que un modelo artificial y simplificado del cerebro de los animales. Una red neuronal artificial es un sistema para el procesamiento de datos, cuya unidad básica de procesamiento está inspirada en la neurona.

En el artículo "A Logical Calculus of the Ideas Inmanent in Nervious Activity" de MacCulloch y Pitts (1943) se describe el primer modelo de redes neuronales. Pocos años después Frank Rosemblat publicó el concepto de perceptrón. La neurona artificial o perceptrón es el bloque de construcción y unidad básica de proceso de una red neuronal artificial. Marvin Minsky, junto con Seymor Papert, escribieron el libro "Perceptrones", donde se establece cómo una sola neurona es incapaz de implementar una función como XOR, y llegaron a que las grandes redes también tienen limitaciones similares. Este concepto limitó el desarrollo de las redes neuronales artificiales por aproximadamente diez años. Luego se retomó el tema y tuvo un primer florecimiento a finales de los 80s y comienzos de los 90s. A finales de los 90s disminuyo nuevamente el interés, pues se consideró que solo resolvían problemas de juguete. En 2005, G. Hinton, premio Nobel de Física 2024, con una red neuronal profunda pudo reconocer dígitos escritos a mano con un 98% de precisión. Los avances han continuado y en 2018, se desarrolló el Transformer, una arquitectura de red, que permitió producir las herramientas de IA generativa, lo que dio la oportunidad a todos los seres humanos de entrar en el mundo de la IA. De manera que las redes neuronales son la columna vertebral de todos estos nuevos desarrollos.

5.2 Modelos de Neuronas

5.2.1 Neurona Natural

La neurona natural, de manera muy simplificada para nuestro interés, consta de un núcleo, un cuerpo denominado soma, al cual están conectadas unas fibras nerviosas llamadas dendritas, fibras terminales de entrada, y un axón, que es una fibra delgada y larga que se divide en su extremo en una serie de fibras terminales de salida.

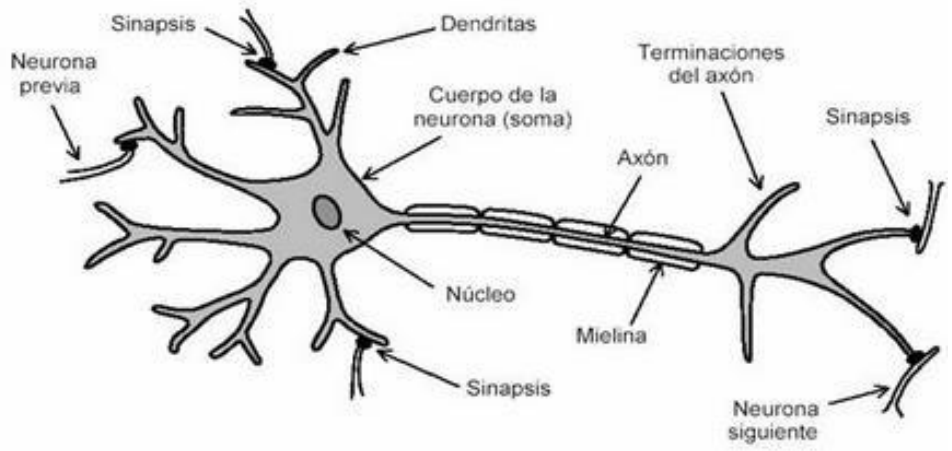


Figura 5.2. Esquema de una neurona natural.

La neurona recibe señales de entrada, provenientes de otras células nerviosas, por las dendritas de la neurona. Las neuronas reciben impulsos de los axones de otras neuronas a través de una conexión llamada sinapsis. La sinapsis (del griego “sin”, que significa “con” y “haptenia”, que significa “con firmeza”) es una conexión más fuerte o más débil dependiendo de la importancia de la señal que llega de la otra neurona. Las señales a través de las cuales se permite la captación y el proceso de control y coordinación de los órganos son de tipo eléctrico. El axón es el encargado de conducir el impulso nervioso desde el soma de una neurona hacia otras células nerviosas.

Las neuronas transmiten su señal eléctrica a consecuencia de un cambio transitorio en la permeabilidad de la membrana plasmática. Cuando el potencial de una membrana, es decir, la suma de todas las señales de entrada se despolariza más allá de un umbral (de 65mV a 55mV), la célula dispara un potencial de acción que se envía a través del axón, como señal de salida de la neurona. El ser humano tiene en promedio 8.6^{10} neuronas y en promedio cada neurona tiene 6.000 conexiones sinápticas, conformando una red con 5.2^{14} enlaces, algo realmente impresionante.

5.2.2 NEURONA ARTIFICIAL

Es un modelo matemático, planteado en términos vectoriales que recibe una serie de señales (valores) $\boldsymbol{X}$ de entrada, que representa el conjunto de dendritas, ponderadas por un peso $\boldsymbol{W}$, que representa el conjunto de sinapsis, y produce una señal de salida $\boldsymbol{Z}$, que representa el potencial de salida del axón. La variable representa el umbral **θ** o el mínimo potencial a partir del cual se dispara el axón. Figura 5.3.

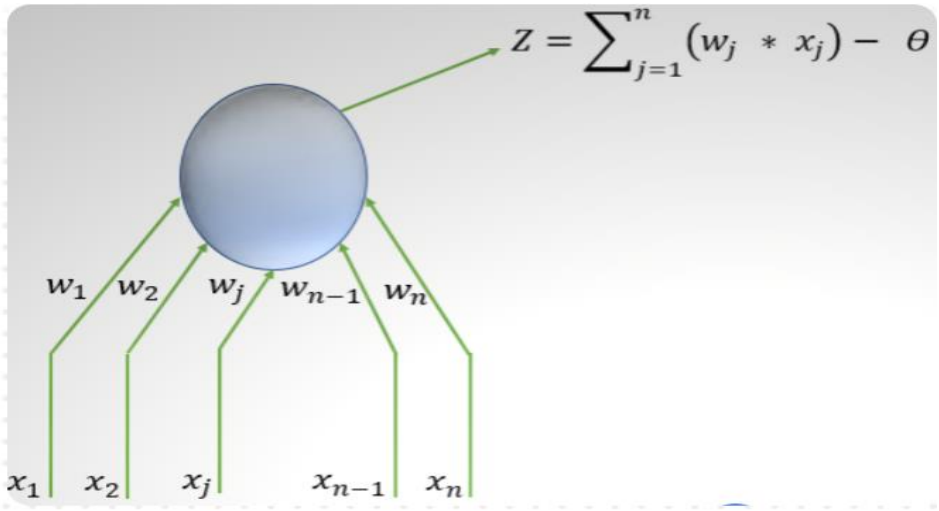
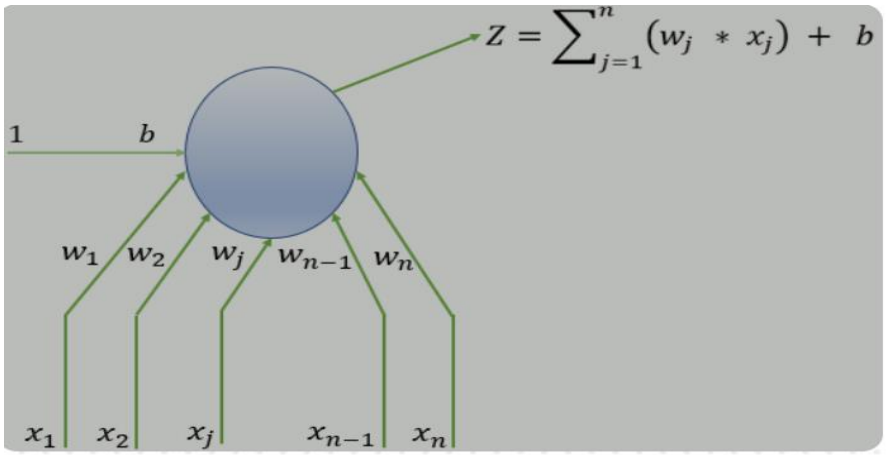


Figura 5.3 Neurona Artificial, modelo abstracto de una neurona natural

Donde

$$Z = -\theta + (w_1\, x_1 + w_2\, x_2 +, \dots, w_j\, x_j +, \dots, +w_n\, x_n) = \sum_{j=1}^{n} w_j x_j - \theta, \quad (1)$$

Para propósitos de notación, hacemos $\theta = -b$ (bias).Esta notación es muy importante ya que nos permite trabajar todo el desarrollo matemático en forma vectorial. Entonces tenemos que:



$$z = \sum_{j=1}^{n} w_j x_j + b$$

$$z = b * 1 + (w_1 x_1 + w_2 x_2 + \cdots + w_n x_n) \ (2),$$

Figura 5.4 Neurona Artificial, modelo abstracto de una neurona natural, con sesgo b.

En términos de aprendizaje de máquina, vamos a denominar el vector X, como el vector de rasgos o **características** (features) y el vector W como el vector de pesos.

$$W = \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_n \end{bmatrix} \quad X = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Dado, que la respuesta del axón es una especie de onda, para acercarnos a esta respuesta, definimos la función de activación $g(Z)$, que es la respuesta que realmente nos interesa. Si se observa la ecuación (2), es la ecuación de una recta en un espacio n-dimensional. La función de activación es la que introduce complejidad a la red.

Veamos la función de activación de la figura 5.5, es la función paso unitario, que es una combinación lineal de los valores de entrada **X** y los pesos **W**. Donde **Z** si es mayor que un umbral determinado, toma el valor 1, sino toma el valor 0.

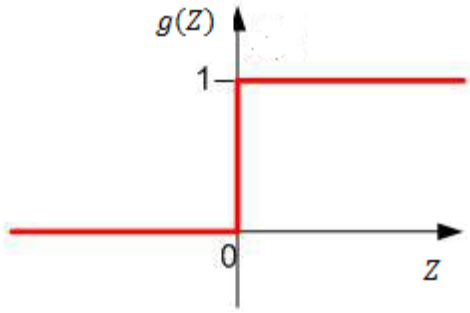


Figura 5.5 Función paso unitario

Sin embargo, esta función tiene varios defectos, en primer lugar, en la naturaleza nunca se presentan cambios en tiempo cero y además es una función que no es derivable, algo bien importante para el algoritmo de aprendizaje. De manera que se utilizan varias funciones de activación que son derivables, la más común es la función de activación sigmoidal, figura 5.6.

$$\varrho(z) \equiv \frac{1}{1+e^{-z}}$$

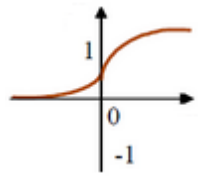


Figura 5.6 Función sigmoide

La función de activación sigmoidal tiene la ventaja adicional que: $\sigma' = \sigma(1-\sigma)$. Es importante recordar que la introducción de la función de activación le da la complejidad a la red neuronal, sin la función de activación, las redes neuronales son simplemente combinaciones lineales.

Aunque la función de activación más usada y más popular ha sido la función sigmoide, la función de activación ReLU (Rectified LinearActivation), se usa en RNAs profundas, porque es más eficiente en el proceso de entrenamiento

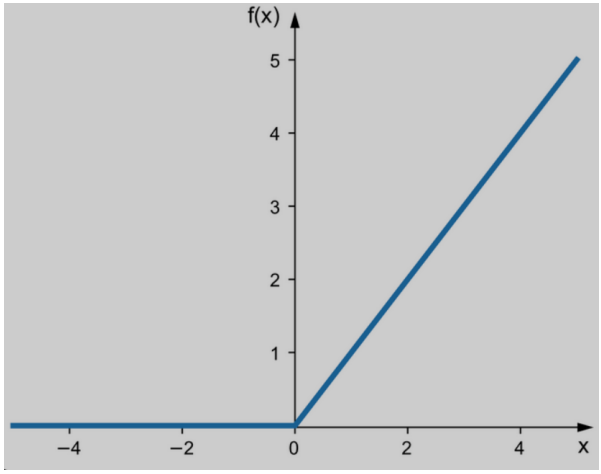
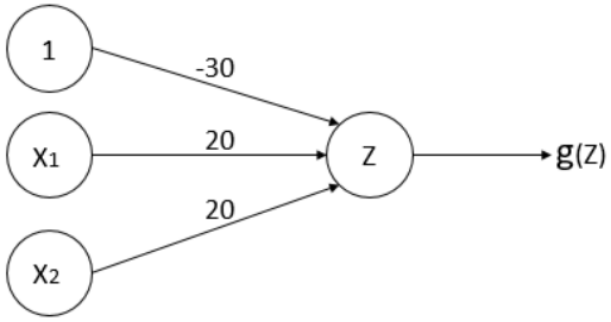


Figura 5.7 Función ReLu

5.3 Ejemplos de clasificación.
Usando la neurona artificial, con la función de activación sigmoidal.
Veamos la compuerta AND, figura 5.7.

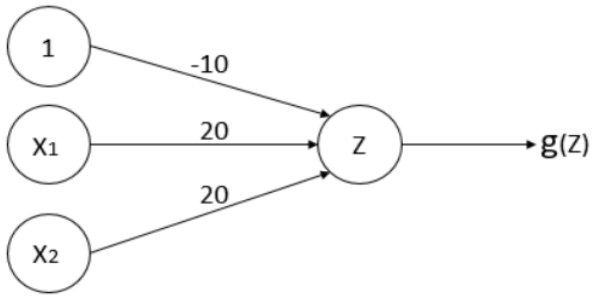
Con los pesos 30, 20 y 20, se calcula la salida $g(Z)$,



X1	X2	W	Z	g(Z)
0	0	-30	-30	~0
0	1	20	-10	~0
1	0	20	-10	~0
1	1		10	~1

Figura 5.7 Red para clasificar una compuerta AND, topología, tabla con vector de pesos.

Ahora la compuerta OR, figura 5.8.
Ahora con los pesos -10, 20 y 20, se calcula $g(Z)$,



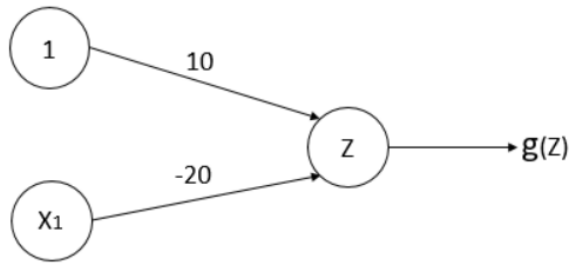
X1	X2	Z	g(Z)	
0	0	-10	~0	w
0	1	10	~1	-10
1	0	10	~1	20
1	1	30	~1	10

Figura 5.8 Red para clasificar una compuerta OR, topología, tabla con vector de pesos.

Como se puede observar ambas redes tienen el mismo número de entradas, solo cambia el peso en la primera entrada, aunque el primero clasifica una compuerta AND, el segundo clasifica una

compuerta OR. El cambio de peso en el vector de sesgo, cambia la función de la red neuronal. ¡Los pesos definen el objetivo de la red neuronal!

Veamos ahora la compuerta NOT, figura 5.9. $Z = (10 - 20x_1)$ y luego se calcula $g(Z)$



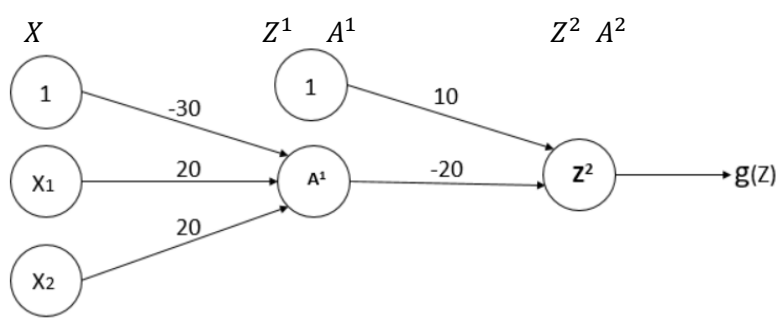
X1	W	Z	g(Z)
0	10	10	~1
1	-20	-10	~0

Figura 5.9 Red para clasificar una compuerta NOT, topología, tabla con vector de pesos

Aquí tenemos una clasificación binaria, $Y = [1, 0]$

Ahora, con base en las conformaciones anteriores vamos a armar la red para la clasificación de una compuerta NAND.

$Z^1 = (-30 + 20x_1 + 20x_2)$, luego se calcula $A^1 = g(Z^1)$; luego $Z^2 = (10a_1 - 20a_2)$ y $A^2 = g(Z^2)$;



X_1	X_2	W^1	Z^1	A^1	W^2	Z^2	g(Z)
0	0	-30	-30	~0	10	10	~1
0	1	20	-10	~0	-20	-10	~0
1	0	20	-10	~0		-10	~0
1	1		10	~1		-30	~0

Figura 5.10 Red para clasificar una compuerta NAND, topología y tabla con matriz de pesos

Ahora tenemos dos capas. En la primera capa obtenemos un valor que hemos llamado Z^1 que se calcula con el vector de pesos W^1 y luego aplicando la función sigmoide obtenemos como A^1 resultado intermedio. En la capa 2, obtenemos un valor que hemos llamado Z^2 que se calcula con el vector de pesos W^2 y luego aplicando la función sigmoide obtenemos el resultado $g(Z^2)$ que corresponde a la compuerta NAND. Como podemos observar ahora tenemos una matriz de pesos W.

Podemos hacer lo mismo para obtener la compuerta NOR, uniendo las conformaciones OR y NOT como se hizo en el caso anterior. Pero también podemos unir las redes de clasificación de las compuertas AND y NOR, que nos da la siguiente topología, figura 5.1

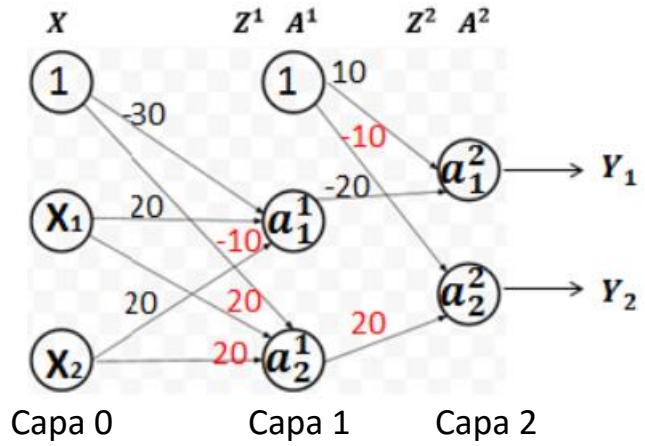


Figura 5.11. Red para clasificar las compuertas AND y OR.

$z_1^1 = (-30 + 20x_1 + 20x_2)$ y $z_2^1 = (-10 + 20x_1 + 20x_2)$; calculamos $a_1^1 = g(z_1^1)$ y $a_2^1 = g(z_2^1)$, que son la salida de la capa 1. Para la capa 2, calculamos $z_1^2 = (10 - 20a_1^1)$ y $z_2^2 = (10 + 20a_2^1)$ y finalmente obtenemos $y_1 = a_1^2 = g(z_1^2)$ y $y_2 = a_2^2 = g(z_2^2)$

Los superíndices nos indican la capa, así, el vector $Z^1 = \begin{bmatrix} z_1^1 \\ z_2^1 \end{bmatrix}$, el vector $A^1 = \begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix}$, que son las salidas de la capa 1. La matriz de pesos de la capa 1 es $W^1 = [w_1^1\ w_2^1]$, $B^1 = [b_1^1\ b_2^1]$ y $W^2 = [w_1^2\ w_2^2], B^2 = [b_1^2\ b_2^2]$:

$$W^1 = \begin{matrix} w_1^1 & w_2^1 \\ 20 & 20 \\ 20 & 20 \end{matrix} \quad B^1 = \begin{matrix} -30 & 20 \end{matrix} \quad W^2 = \begin{matrix} w_1^2 & w_2^2 \\ -20 & -20 \end{matrix} \quad B^2 = \begin{matrix} 10 & -10 \end{matrix}$$

En este caso tenemos una multi-clasificación, es decir $g(Z^2)$ es un vector. En términos de clasificación. Si k es el número de clases, $k = 2$, en este caso tenemos dos clases NAND y NOR.

$$Y = \begin{array}{|c|c|} \hline Y1 & Y2 \\ \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}$$

Si por ejemplo estamos clasificando números arábigos, tendríamos $k = 10$.

Las redes neuronales artificiales están diseñadas para procesar información y aprender patrones complejos a partir de los datos. Sus funciones principales son la clasificación y la predicción.

Clasificación

Se puede definir de manera general la clasificación como el agrupamiento de cosas que comparten ciertas características y cualidades. Por ejemplo, clasificar formas geométricas, números escritos a mano, o simplemente huevos. En IA, la clasificación es el problema de identificar a que conjunto de categorías pertenece una nueva observación, con base en un proceso computacional en el que se utiliza un modelo algorítmico y en el que se ha tenido un conjunto de datos de entrenamiento que contiene observaciones cuya membresía de categorías se conoce.

Predicción

Con base en datos históricos, datasets, las redes neuronales, pueden predecir o estimar futuros valores numéricos en un determinado horizonte de tiempo. Es importante ser conscientes de que siempre estas predicciones tienen una incertidumbre inherente.

5.4 Aprendizaje Supervisado

Uno de los procesos más importantes en RNA es el proceso de aprendizaje. El aprendizaje puede hacerse de manera supervisada o no supervisada. En el capítulo siguiente volveremos sobre este tema. En el aprendizaje o entrenamiento supervisado, se proveen tanto las entradas, las características, como las salidas, o rótulos, correspondientes a esas entradas. Un atributo, o característica, es un aspecto específico de un objeto, o del rótulo. En ML se supone que el conjunto de características describe un objeto.

La red entonces procesa las entradas y compara sus salidas resultantes contra las salidas deseadas o rótulos. En la mayoría de los casos no corresponden y se debe calcular su diferencia o error. Entonces ahora se busca ajustar los pesos que controlan la red para disminuir el error, hasta que de acuerdo con un determinado criterio se considera que el error es aceptable.

Generalización del aprendizaje

Se busca que con la red neuronal se pueda, por ejemplo, predecir apropiadamente el costo de cualquier inmueble, es decir la red debe generalizar el concepto para cualquier tipo de inmueble. Se pueden presentar dos problemas, que el modelo sea demasiado general o que sea demasiado específico. Esto se debe tener en cuenta al entrenar la red.

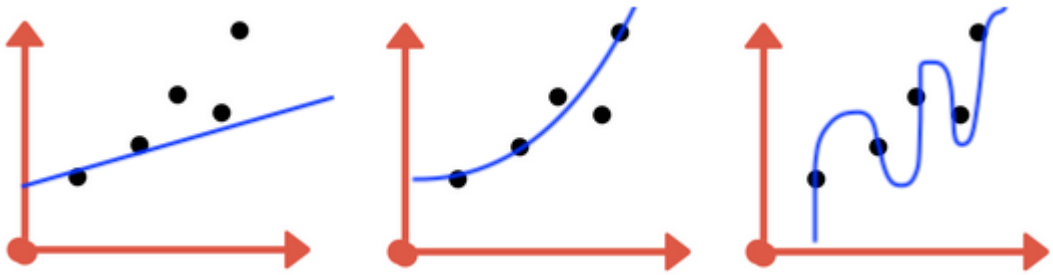


Figura 5.12. En la figura de la izquierda se tiene sub-entrenamiento, también conocido como underfitting; en el centro, el entrenamiento adecuado, y en la figura de la derecha se tiene sobre-entrenamiento o overfitting.

Para solucionar este problema, el dataset se divide en tres conjuntos, el conjunto de entrenamiento, generalmente del 80% del dataset; el conjunto de prueba, con el que se busca obtener una estimación del comportamiento de la red en el mundo real; y el conjunto de validación, que nos permite una evaluación de la generalización del modelo.

Propagación

Las redes que hemos visto hasta ahora se conocen como feedforward, cuya característica es que el proceso va desde la capa de entrada hasta capa de salida. No hay retroalimentación hacia atrás. Es la red más simple y utilizada. Entonces, vamos a generalizar una red feed-forward, con su nomenclatura, con base en la figura 5.13.

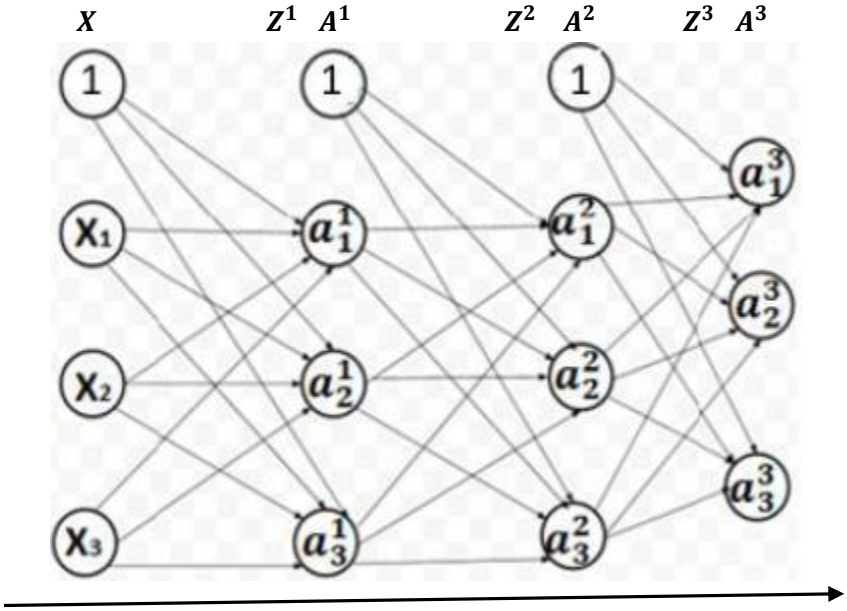


Figura 5.13. Red de tres capas para mostrar la propagación, desde el vector de entrada.

Vamos a describir cómo se propagan los datos en la red desde la entrada.

Capa 1:

$$z_1^1 = b_1^1 + (w_{11}^1 x_1 + w_{21}^1 x_2 + w_{31}^1 x_3), \quad a_1^1 = g\,(z_1^1)$$

$$z_2^1 = b_2^1 + (w_{12}^1 x_1 + w_{22}^1 x_2 + w_{32}^1 x_3), \quad a_2^1 = g\,(z_2^1)$$

$$z_3^1 = b_3^1 + (w_{13}^1 x_1 + w_{23}^1 x_2 + w_{33}^1 x_3), \quad a_3^1 = g\,(z_3^1)$$

con

$$W_T^1 = B^1 + W^1 = \begin{matrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{matrix} + \begin{matrix} w_{11}^1 & w_{21}^1 & w_{31}^1 \\ w_{12}^1 & w_{22}^1 & w_{32}^1 \\ w_{13}^1 & w_{23}^1 & w_{33}^1 \end{matrix}$$

Matriz de pesos de la capa 1

Matricialmente, $Z^{(1)} = B^1 + \left(W^{(1)T}\right)X$ y $A^{(1)} = g(Z^{(1)})$; para generalizar podemos decir que $A^{(0)} = X$, y así la ecuación matricial para la capa 1 sería $Z^{(1)} = B^1 + (W^{(1)T})A^{(0)}$.

Capa 2:

$$z_1^2 = b_1^2 + (w_{11}^2 a_1^1 + w_{21}^2 a_2^1 + w_{31}^2 a_3^1), \quad a_1^2 = g\,(z_1^2)$$

$$z_2^2 = b_2^2 + (w_{12}^2 a_1^1 + w_{22}^2 a_2^1 + w_{32}^2 a_3^1), \quad a_2^2 = g\,(z_2^2)$$

$$z_3^2 = b_3^2 + (w_{13}^2 a_1^1 + w_{23}^2 a_2^1 + w_{33}^2 a_3^1), \quad a_3^2 = g\,(z_3^2)$$

con

$$W_T^2 = B^2 + W^2 = \begin{matrix} b_1^2 \\ b_2^2 \\ b_3^2 \end{matrix} + \begin{matrix} w_{11}^2 & w_{21}^2 & w_{31}^2 \\ w_{12}^2 & w_{22}^2 & w_{32}^2 \\ w_{13}^2 & w_{23}^2 & w_{33}^2 \end{matrix}$$

Matriz de pesos de la capa 2.

Matricialmente $Z^{(2)} = B^2 + (W^{(2)T})A^{(1)}$

Capa 3:

$$z_1^3 = b_1^3 + (w_{11}^3 a_1^2 + w_{21}^3 a_2^2 + w_{31}^3 a_3^2), \quad a_1^3 = g\,(z_1^3)$$

$$z_2^3 = b_2^3 + (w_{12}^3 a_1^2 + w_{22}^3 a_2^2 + w_{32}^3 a_3^2), \quad a_2^3 = g\,(z_2^3)$$

$$z_3^3 = b_3^3 + (w_{13}^3 a_1^2 + w_{23}^3 a_2^2 + w_{33}^3 a_3^2), \quad a_3^3 = g\,(z_3^3)$$

$$W_T^3 = B^3 + W^3 = \begin{matrix} b_1^3 \\ b_2^3 \\ b_3^3 \end{matrix} + \begin{matrix} w_{11}^3 & w_{21}^3 & w_{31}^3 \\ w_{12}^3 & w_{22}^3 & w_{32}^3 \\ w_{13}^3 & w_{23}^3 & w_{33}^3 \end{matrix}$$

Matriz de pesos de la capa 3

Matricialmente
$Z^{(3)} = B^3 + (W^{(3)T})A^{(2)}$ y $A^{(3)} = g(Z^{(3)})$. Generalizando, para cualquier capa oculta i, se tiene que:

$$Z^{(i)} = B^{(i)} + (W^{(i)T})A^{(i-1)} \text{ y } A^{(i)} = g(Z^{(i)}).$$

Al analizar esta ecuación, encontramos que se pasa de una capa a la otra con la ecuación de la izquierda y el vector de salida $A^{(i)}$ se encuentra aplicando la función de activación a $Z^{(i)}$.

Volviendo a la figura 5.11, como la capa 3 es la última capa de la red, los $A^{(3)}$ son la respuesta de la red que notamos como $H_w(X)$. En otras palabras, el resultado de la propagación del vector X a través de la red, es el vector $H_w(X)$.

Aprendizaje supervisado
Con el fin de encontrar los pesos apropiados para que la red responda apropiadamente a los patrones que queremos, debemos trabajar en el ajuste de los pesos. Para entrenar la red en esos patrones hacemos ***aprendizaje supervisado***, proveyendo, para este proceso un conjunto de entrenamiento: $(X_{(1)}, Y_{(1)})$, $(X_{(2)}, Y_{(2)})$, … , $(X_{(m)}, Y_{(m)})$. Donde los vectores X son las características (features) o patrones de entrada y los vectores Y son los patrones que debe reconocer o las clasificaciones que debe producir la red.

Analicemos un poco el conjunto de entrenamiento, por ejemplo el conjunto de entrenamiento de MNIST que consiste en dígitos escritos a mano de 10.000 ejemplos. Los dígitos son imágenes de un tamaño normalizado de 28 X 28 pixeles, 784 pixeles, en escala de grises, con un rótulo que dice que número es, de manera que los valores de los rótulos son de 0 a 9. De manera que en el par $(X_{(m)}, Y_{(m)})$, el $X_{(m)}$ corresponde a un vector de 748 valores y $Y_{(m)}$corresponde a un rótulo entre 0 y 9. Así, en la figura 5.15, los valores en el recuadro son los corresponden a los 784 valores de los pixeles el número escrito a mano y la flecha me indica su rótulo

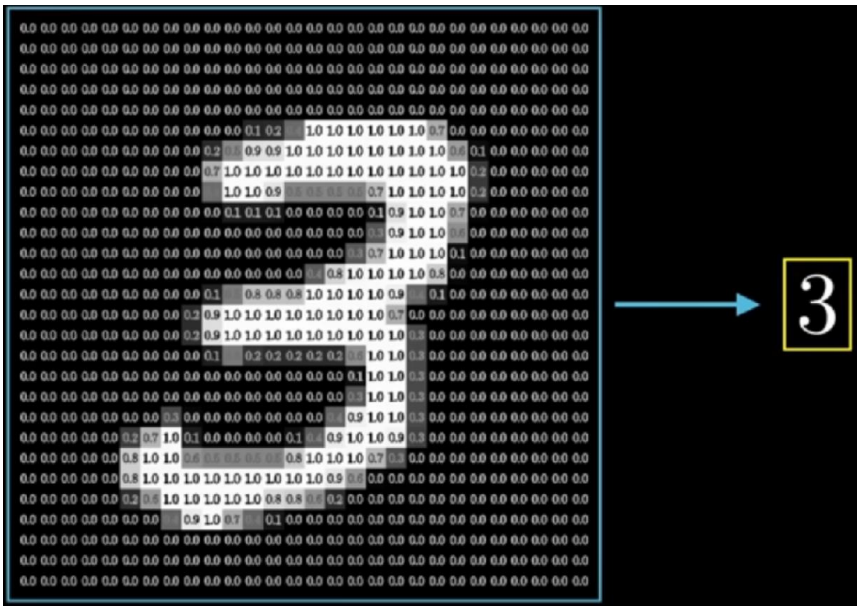


Figura 5.14. El número 3 escrito a mano, normalizado como una imagen de 28 x 28, 784 pixeles, y su rótulo correspondiente, 3.

Se trata entonces de darle como entrada a una red neuronal los 784 pixeles y un rótulo; pero 10.000 veces, para que se obtengan los pesos que permitan clasificar cualquier número escrito a mano, normalizado a 28 X 28 pixeles, ver figura 5.15.

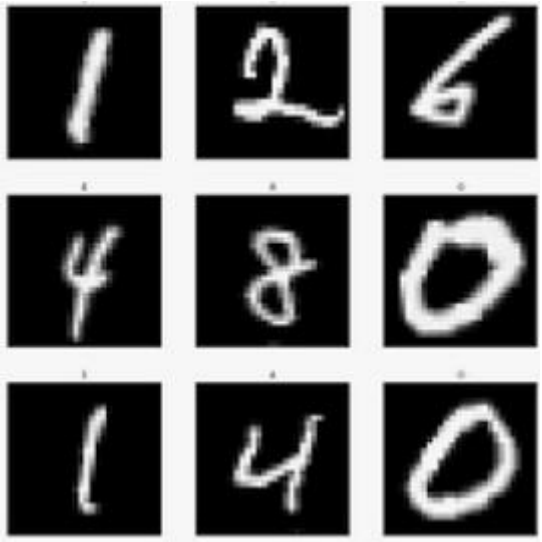


Figura 5.15. En la figura aparecen los números escritos a mano como imágenes de 28 X 28, 784 pixeles, valores de X, aparte vienen los valores de los rótulos 1, 2, 6, 4, 8, 0, 1, 4, 0; que corresponden a los valores de Y, o rótulos.

5.5 Algoritmo de aprendizaje y función de pérdida

La topología de la red depende del número de características (features) del vector de entrada, del número de salidas en la capa de salida, que corresponde a las clasificaciones

que hace la red o del resultado que se espera y del número de capas ocultas, que generalmente es de dos o tres capas. Además, todas las neuronas de la capa i se conectan con todas las neuronas de la capa $i\ +1$; grafos completamente conectados

Inicialmente se asignan valores aleatorios entre [0, 1] a los pesos de la red, aunque algunos autores recomiendan valores menores de 0.5. Una vez se hace la propagación a través de la red de los vectores X, con esos valores de pesos, se obtiene el vector de salida Y. La diferencia con el rotulo de entrenamiento Y_r, es $e = |Y_r - Y|$, es el error de clasificación de la red $e = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix}$

El proceso de entrenamiento o aprendizaje consiste en hacer que ese error sea menor que un Ɛ definido. También se define la función de pérdida como el error promedio E como:

$$E = \frac{1}{2}\,(e)^2 \;=\; \frac{1}{2}\,(e)^T e$$

La idea es minimizar el error haciendo que se afecten todos los pesos en cada una de las capas de la red. Como solo se conoce el error en la última capa, se comienza minimizando el error en esa capa, ajustando el vector de sesgo y la matriz de peso de esa capa. Y luego se sigue hacia las capas internas, hasta llegar a la capa de entrada. Esto es lo que se conoce como algoritmo de retropropagación o Backpropagation.

Backpropagation.

Tercera capa. Dado que tenemos 3 capas, se minimiza el error con respeto a los pesos en la capa 3, en este caso la capa de salida, ver figura 5.16. Siguiendo la regla de la cadena se tiene:

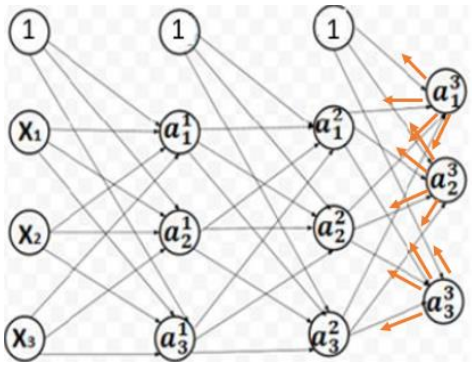


Figura 5.16, Con base en el error promedio E, se ajustan lo valores de la matriz de pesos de la última capa.

De manera que se busca encontrar el cambio en el error promedio E con base en el cambio en la matriz de pesos W_T^3 : $\frac{\partial E}{\partial W_T^3}$. Pero lo que conocemos es Z^3, de manera que hacemos:

$$\frac{\partial E}{\partial W_T^3} = \frac{\partial E}{\partial Z^3}\,\frac{\partial Z^3}{\partial W_T^3}$$

O También:

$\frac{\partial E}{\partial W_T^3} = \frac{\partial Y}{\partial Z^3} \frac{\partial e}{\partial Y} \frac{\partial E}{\partial e} \frac{\partial Z^3}{\partial W_T^3}$, recordando que $A^3 = Y = g(Z^3)$. $Y = \sigma(Z^3)$

Es importante recordar que $\sigma' = \sigma(1 - \sigma)$

Así, $\frac{\partial Y_i^3}{\partial Z_j^3} = \begin{cases} 1, si\ i = j \\ 0, si\ i \neq j \end{cases}$ o $\frac{\partial Y^3}{\partial Z^3} = \begin{bmatrix} \sigma'(Z_1^3) & 0 & 0 \\ 0 & \sigma'(Z_2^3) & 0 \\ 0 & 0 & \sigma'(Z_3^3) \end{bmatrix}$ y notando

Nos queda que: $\frac{\partial Y}{\partial Z^3} = diag\left(\sigma'(Z^3)\right)$. Recordando $e = |Y_r - Y|$

Se tiene, $\frac{\partial e}{\partial Y} = \frac{\partial e_i}{\partial Y_j} = \begin{cases} -1, si\ i = j \\ 0, si\ i \neq j \end{cases}$ o $\frac{\partial e_i}{\partial Y_j} = -\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1) \end{bmatrix}$ = - 1

1

Ahora, $\frac{\partial E}{\partial e} = \frac{\partial}{\partial e}\left(\frac{1}{2}\ (e)^T e\right) = e$. Recordando $Z^{(3)} = B^{(3)} + (W^{(3)T})A^{(2)}$

Ahora, $\frac{\partial Z^3}{\partial W_T^3} = \frac{\partial}{\partial W_T^3}(B^{(3)} + (W^{(3)T})A^{(2)})$ 0, $\frac{\partial Z^3}{\partial W_T^3} = 1 + A^{(2)T}$

Reuniendo todos los términos:

$$\frac{\partial E}{\partial W_T^3} = \frac{\partial Y^3}{\partial Z^3} \frac{\partial e}{\partial Y^3} \frac{\partial E}{\partial e} \frac{\partial Z^3}{\partial W_T^3}$$

$$\frac{\partial E}{\partial W_T^3} = diag\left(\sigma'(Z^3)\right) - 1\ e\ (1 + A^{(2)T})$$

$$\frac{\partial E}{\partial W_T^3} = -diag\left(\sigma'(Z^3)\right)\ e\ (1 + A^{(2)T})$$

Haciendo $\delta^3 = diag\left(\sigma'(Z^3)\right) e$, entonces

$\frac{\partial E}{\partial W_T^3} = -\delta^3(1 + A^{(2)T})$ o $\frac{\partial E}{\partial W_T^3} = -\delta^3 - \delta^3 A^{(2)T}$

donde $\frac{\partial E}{\partial W_T^3} = \frac{\partial E}{\partial B^3} + \frac{\partial E}{\partial W^3}$

con $\frac{\partial E}{\partial B^3} = -\delta^3$ y $\frac{\partial E}{\partial W^3} = -\delta^3 A^{(2)T}$

De manera que con estos valores ajustamos lo pesos de la última capa.

Segunda capa. Ahora se minimiza el error promedio E con respeto a los pesos de la capa 2, ver figura 5.17. Siguiendo la regla de la cadena se tiene:

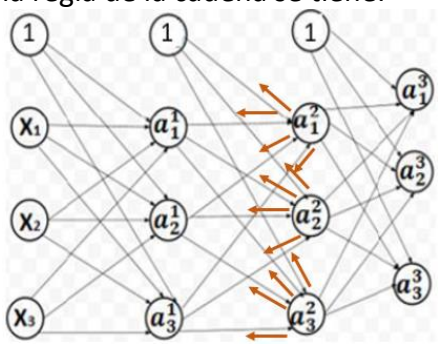


Figura 5.17. Con base en el error promedio E, se ajustan lo valores de la matriz de pesos de la segunda capa.

$$\frac{\partial E}{\partial W_T^2} = \frac{\partial E}{\partial Z^2}\frac{\partial Z^2}{\partial W_T^2}$$

$$\frac{\partial E}{\partial W_T^2} = \frac{\partial A^2}{\partial Z^2}\frac{\partial Z^3}{\partial A^2}\frac{\partial E}{\partial Z^3}\frac{\partial Z^2}{\partial W_T^2}$$

Ahora, $\frac{\partial A_i^2}{\partial Z_j^2} = \begin{cases} 1, si\ i = j \\ 0, si\ i \neq j \end{cases}$ o $\frac{\partial A^2}{\partial Z^2} = \begin{bmatrix} \sigma'(Z_1^2) & 0 & 0 \\ 0 & \sigma'(Z_2^2) & 0 \\ 0 & 0 & \sigma'(Z_3^2) \end{bmatrix}$

$$\frac{\partial A^2}{\partial Z^2} = diag(\sigma'(Z^2))$$

Ahora, $\frac{\partial Z^3}{\partial A^2} = \frac{\partial}{\partial W_T^3}(B^{(3)} + (W^{(3)T})A^{(2)})$ o, $\frac{\partial Z^3}{\partial A^2} = W^{(3)T}$

Ahora, para calcular $\frac{\partial E}{\partial Z^3}$ Teníamos que: $E = \frac{1}{2}\ (e)^2$ o $E = \frac{1}{2}\ (Y_r - Y)^2$ o $E = \frac{1}{2}\ (Y_r - g(Z^{(3)}))^2$ ahora, derivando con respecto a $Z^{(3)}$, se tiene:

$\frac{\partial E}{\partial Z^3} = \left(Y_r - g\left(Z^{(3)}\right)\right) - diag\left(\sigma'(Z^3)\right)$, pero $e = |Y_r - g\left(Z^{(3)}\right)|$, entonces reacomodando

$\frac{\partial E}{\partial Z^3} = -diag\left(\sigma'(Z^3)\right) e$, y se había definido $\delta^3 = diag\left(\sigma'(Z^3)\right) e$

$$\frac{\partial E}{\partial Z^3} = -\delta^3$$

Ahora, $\frac{\partial Z^2}{\partial W_T^2} = \frac{\partial}{\partial W_T^2}\left(B^{(2)} + W^{(2)T}A^1\right)$ o, $\frac{\partial Z^2}{\partial W_T^2} = 1 + A^{(1)T}$

Reuniendo los términos

$$\frac{\partial E}{\partial W_T^2} = -diag\left(\sigma'(Z^2)\right) W^{(3)T}\ \delta^3\ (1 + A^{(1)T})$$

Haciendo $\delta^2 = diag\left(\sigma'(Z^2)\right) W^{(3)T}\ \delta^3$, entonces

$\frac{\partial E}{\partial W_T^2} = -\delta^2\left(1 + A^{(1)T}\right)$ o $\frac{\partial E}{\partial W_T^2} = -\delta^2 - \delta^2 A^{(1)T}$

Donde $\frac{\partial E}{\partial B^2} = -\delta^2$ y $\frac{\partial E}{\partial W^2} = -\delta^2 A^{(2)T}$

$$\frac{\partial E}{\partial W_T^3} = \frac{\partial E}{\partial B^3} + \frac{\partial E}{\partial W^3}$$

Primera capa. Ahora se minimiza el error promedio E con respeto a los pesos de la capa 1, ver figura 5.18. Siguiendo la regla de la cadena se tiene:

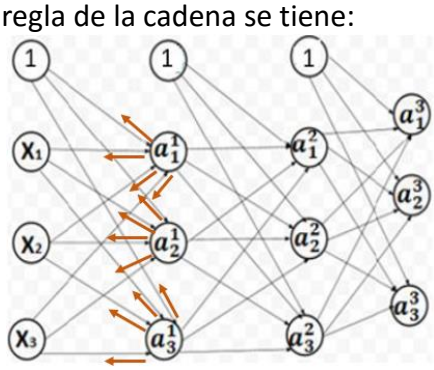


Figura 5.18. Con base en el error promedio E, se ajustan lo valores de la matriz de pesos de la primera capa.

$$\frac{\partial E}{\partial W_T^1} = \frac{\partial E}{\partial Z^1}\frac{\partial Z^1}{\partial W_T^1}$$

$$\frac{\partial E}{\partial W_T^1} = \frac{\partial A^1}{\partial Z^1}\frac{\partial Z^2}{\partial A^1}\frac{\partial E}{\partial Z^2}\frac{\partial Z^1}{\partial W_T^1}$$

Ahora, $\frac{\partial A_i^1}{\partial Z_j^1} = \begin{cases} 1, si\ i = j \\ 0, si\ i \neq j \end{cases}$ o $\frac{\partial A_i^1}{\partial Z_j^1} = \begin{bmatrix} \sigma'(Z_1^1) & 0 & 0 \\ 0 & \sigma'(Z_1^1) & 0 \\ 0 & 0 & \sigma'(Z_1^1) \end{bmatrix}$

$$\frac{\partial A^1}{\partial Z^1} = diag(\sigma'(Z^1))$$

Ahora, $\frac{\partial Z^2}{\partial A^1} = \frac{\partial}{\partial A^1}(B^{(2)} + W^{(2)T}A^1)$ 0 $\frac{\partial Z^2}{\partial A^1} = W^{(2)T}$

Ahora, para calcular $\frac{\partial E}{\partial Z^2}$ Teníamos que: $E = \frac{1}{2}\ (e)^2$ o $E = \frac{1}{2}\ (Y - Y_c)^2$ o $E = \frac{1}{2}\left(Y - g\left(Z^{(3)}\right)\right)^2$, reemplazando $Z^{(3)}$ por $B^3 + (W^{(3)T})A^{(2)}$ se tiene:
$E = \frac{1}{2}\left(Y - g\left(B^3 + (W^{(3)T})A^{(2)}\right)\right)^2$, pero $A^{(2)} = g(Z^{(2)})$; reemplazando:
$E = \frac{1}{2}\left(Y - g\left(B^3 + (W^{(3)T})g(Z^{(2)})\right)\right)^2$ y derivando

$$\frac{\partial E}{\partial Z^2} = \left(Y - g\left(Z^{(3)}\right)\right)\ \left(-diag\left(\sigma'\left(Z^3\right)\right)\right)\ W^{(3)T} diag\left(\sigma'\left(Z^2\right)\right)$$

$\frac{\partial E}{\partial Z^2} = -diag\left(\sigma'\left(Z^3\right)\right)\ e\ W^{(3)T}\ diag\left(\sigma'\left(Z^2\right)\right)$ pero $\delta^3 = diag\left(\sigma'(Z^3)\right)\ e$

$\frac{\partial E}{\partial Z^2} = \delta^3\ W^{(3)T}\ diag\left(\sigma'\left(Z^2\right)\right)$ y $\delta^2 = diag\left(\sigma'(Z^2)\right)\ W^{(3)T}\ \delta^3$

$$\frac{\partial E}{\partial Z^2} = \delta^2$$

Ahora, $\frac{\partial Z^2}{\partial A^1} = \frac{\partial}{\partial A^1}\left(B^{(2)} + W^{(2)T}A^1\right)$,

$\frac{\partial Z^1}{\partial W_T^1} = 1 + A^{(0)T}$

$$\frac{\partial E}{\partial W_T^1} = -diag(\sigma'(Z^1))\ W^{(2)T}\ \delta^2\ (1 + A^{(0)T})$$

Haciendo $\delta^1 = diag(\sigma'(Z^1))\ W^{(2)T}\delta^2$, entonces

$$\frac{\partial E}{\partial W_T^1} = -\delta^1\ (1 + A^{(0)T})$$

Donde $\frac{\partial E}{\partial B^1} = -\delta^1$ y $\frac{\partial E}{\partial W^1} = -\delta^1 A^{(1)T}$

En este caso la operación + nos indica se debe considerar una operación con dos componentes, uno que es la actualización de los pesos que representan el umbral de manera separada y el otro que corresponde a la actualización de la matriz de pesos.

Actualización de los pesos

Los pesos se actualizan gradualmente usando el descenso del gradiente. Con η como la tasa de aprendizaje.

Generalizando, para cualquier capa interna:

$$\delta^k = diag(\sigma'(Z^k))\ W^{(k+1)T}\delta^{k+1}$$

$$\Delta W^{(K+1)} = -\eta\,\delta^k - \eta\,\delta^k\,A^{(k-1)T}$$
$$W^{(K)} = W^{(K)} + \Delta W^{(K+1)}$$

En las fórmulas anteriores, la variable η se conoce como tasa de aprendizaje. Es un valor que se toma entre 0 y 1. Si por ejemplo se toma $\eta = 0.2$, quiere decir que solamente se tiene en cuenta el 20% del error para la actualización de los pesos, se considera como un hiperparámetro de configuración. Los pesos también se consideran hiperparámetros.

Resumiendo, en el aprendizaje por Backpropagation se calcula el error en la salida. Luego se pasa el error hacia atrás y se pondera a lo largo de cada capa, permitiendo calcular los cambios que se deben realizar en la matriz W de pesos y el vector B de sesgos, en cada capa. Cuando se llega a una unidad, se propaga, se multiplica el error propagado hacia atrás por la derivada de la unidad. Es un algoritmo recursivo.

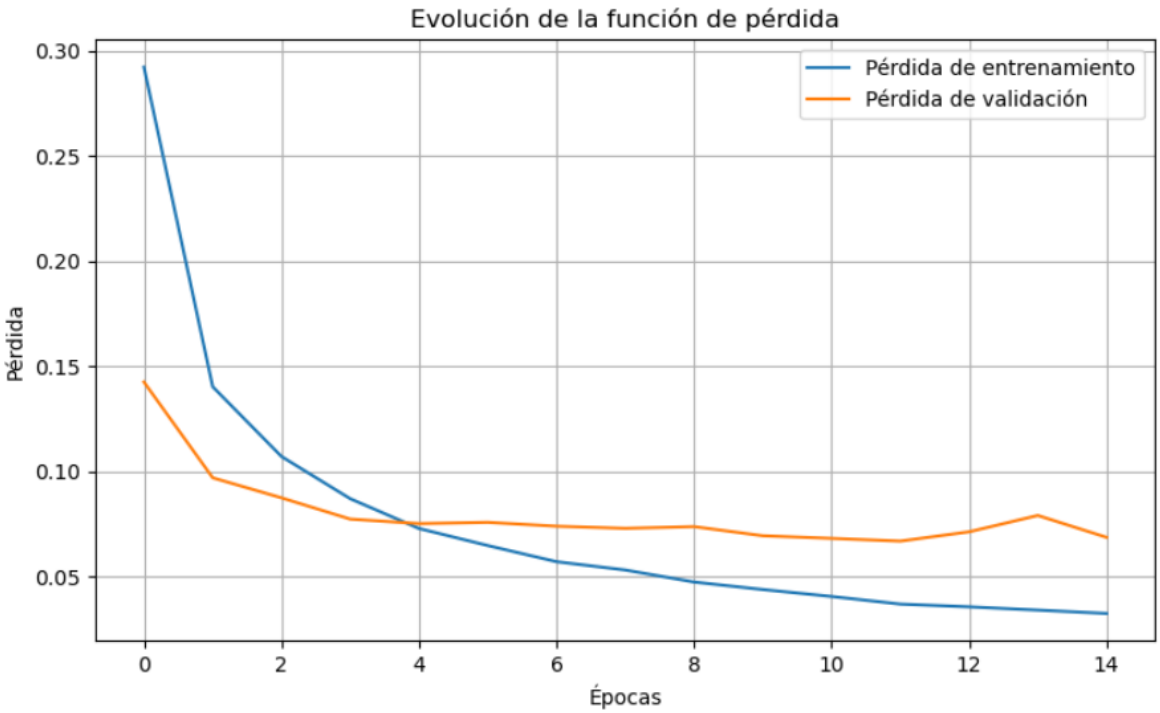
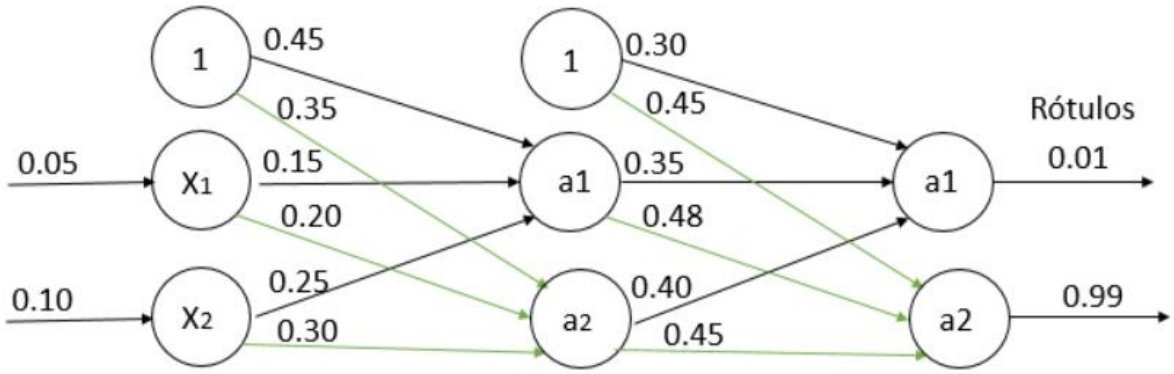


Figura 5.17. Cómo va evolucionando la función de pérdida, con 15 épocas.

El algoritmo de retropropagación fue descrito inicialmente por P. Werbos en 1974, pero G. Hinton y su grupo, mostraron la eficiencia en el entrenamiento de redes multicapa en los 80s.

5.6 Ejemplos

Ejemplo 1. Suponga que tiene la siguiente red neuronal:



Vamos a ver el programa que se está en NoteBook de Jupyter.

```
import numpy as np
```

```
nn_0 = 2
nn_1 = 2
nn_2 = 2

b1 = np.array([[0.45], [0.35]])
W1 = np.array([[0.15, 0.20], [0.25,0.30]]) # Inicialización de los sesgos y los pesos
b2 = np.array([[0.30], [0.45]])
W2 = np.array([[0.35, 0.48], [0.40,0.45]])

Z1 = np.zeros((nn_1,1))
A1 = np.zeros((nn_1,1))
Z2 = np.zeros((nn_2,1))   # Inicialización de los vectores de entrada y salida de las capas
A2 = np.zeros((nn_2,1))
```

Se define la topología de la red, los valores de los sesgos y pesos, de manera arbitraria; y los vectores de propagación de la red, como numéricos.

```
def propaga(Xo):                        # Función de propagación hacia adelante
    Z1 = np.dot(W1,Xo) + b1 # Z1 = W1.Xo + W10
    A1 = sig(Z1)                    # A1 = F(Z1)
    Z2 = np.dot(W2,A1) + b2 # Z2 = W2.A1 + W20
    A2 = sig(Z2)                    # A2 = F(Z2)
    #print("A2 =", A2)
    return A2, Z2, A1, Z1

def sig(s):     # Funcion de activacion sigmoide
    return 1/(1+np.exp(-s))

def dSig(s):
    df = sig(s.T[0])*(sig(s.T[0]))
    ds = np.diag(df)
    return ds
```

Se define la función de propagación, su parámetro es solo el vector de datos de entrada Xo, y hace la propagación a través de la red, con base en los vectores de sesgo y las matrices de pesos.

Encuentra los vectores de resultados en cada capa: Z1, A1, Z2 y A2. Luego viene la función sigma. Y de la derivada de sigma, para el algoritmo de Backpropagation.

```
X = np.array([[0.05], [0.10]])
Yr = np.array([[0.01], [0.99]])
```

X es el vector de entrada a la red o vector de características, con los valores 0.05 Y 0.10. Luego el vector Yr, que es el resultado que queremos que aprenda, es el vector de rótulos.

```
A2, Z2, A1, Z1 = propaga(X)
print(A2)
```

```
[[0.69051105]
 [0.72421384]]
```

Con base en el vector de características X, se encuentran las salidas de cada capa de la red, no interesa la última A2, para contrastarla contra el vector de rótulos Yr.

```
def error(A2, Yd):
    err = Yd - A2
    return err
```

Permite calcular el error entre lo que produce la red, A2 y lo que se espera que debe dar, Yr.

```
def backpropagation(X,Yr,A2, Z2, A1, Z1): # Retropropagación
        delta2 = np.dot(dSig(Z2),error(A2,Yr))
        dEdW2 = -np.dot(delta2,A1.T)                    # componente de corrección para W2
        dEdb2 = -delta2

        delta1 = np.dot(np.dot(dSig(Z1),W2.T),delta2)
        dEdW1 = -np.dot(delta1,X.T)                     # componente de corrección para W1

        dEdb1 = -delta1
        return dEdW2,dEdb2,dEdW1,dEdb1
```

El algoritmo de retropropagación, funcional, cuyas entradas son las salidas de cada una de las capas, el vector de entrada X y el rótulo Yr.

```
dEdW2,dEdb2,dEdW1,dEdb1 = backpropagation(X,Yr, A2, Z2, A1, Z1)
```

Se llama la función de retropropagación y produce los cambios que se deben hacer a los vectore4s de sesgo y a las matrices de pesos.

```
eta = 0.1
```

Definición de la tasa de aprendizaje

```
W2 = W2 - eta * dEdW2
b2 = b2 - eta * dEdb2
W1 = W1 - eta * dEdW1
b1 = b1 - eta * dEdb1
```

Se actualizan los pesos, es la primera iteración o época del aprendizaje.

```
A2, Z2, A1, Z1 = propaga(X)
print(A2)
```

```
[[0.67821878]
 [0.72890846]]
```

Los valores han mejorado, aunque muy poco.

```
i = 0
while i < 50000:
    A2, Z2, A1, Z1 = propaga(X)
    err = error(A2, Yr)

    dEdW2,dEdb2,dEdW1,dEdb1 = backpropagation(X,Yr, A2, Z2, A1, Z1)
    W2 = W2 - eta * dEdW2
    b2 = b2 - eta * dEdb2
    W1 = W1 - eta * dEdW1
    b1 = b1 - eta * dEdb1

    i = i + 1
print(A2)
```

```
[[0.03341429]
 [0.9900029 ]]
```

Es la forma en que se corre el aprendizaje de una red neuronal aquí por 50.000 épocas. El resultado ha mejorado. Se diría que el segundo vector está bien pero falta el otro resultado. Este es el proceso de aprendizaje de una red neuronal.

Ejemplo 2. Reconocimiento de vocales. Se definen vocales de tamaño 4 X 5 pixeles. Se entrena la red y luego se le da una vocal para que se reconozca. Por favor cargar ReconocVocales y correrlo.

De números con base en el data set MNIST de números escritos a mano, hacer reconocimiento de números escritos a mano.

5.7 Aprendizaje profundo (Deep Learning)

Como se puede apreciar en la figura 5.19, en Deep Learning se tiene una RNA, como las que se han estudiado, simplemente que el adjetivo “Deep” hace referencia a que el número de capas ocultas es relativamente alto, hablamos del orden de 7, 8 o 9 capas. Claramente funciona el mismo algoritmo de retropropagación.

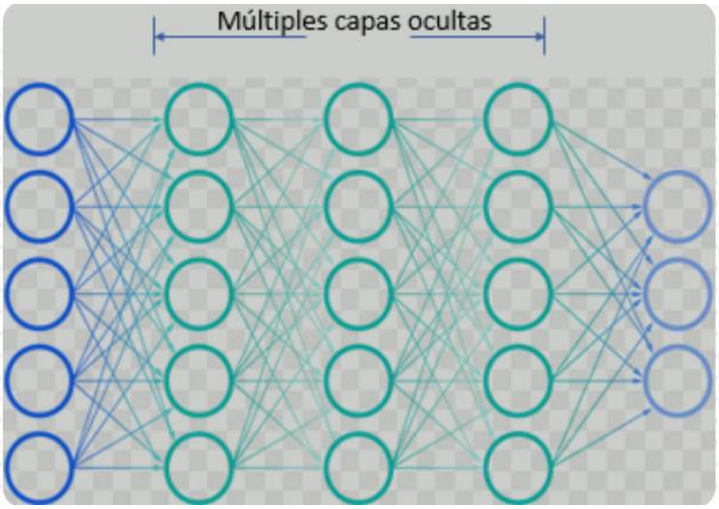


Figura 5.19. Red neuronal profunda. Con varias capas ocultas.

5.8 Hacia Deep learnig, conceptos básicos de computación visual

5.8.1 Redes convolucionales.

Una convolución es simplemente un filtro de pesos que se usa para multiplicar un pixel por sus vecinos, para obtener un nuevo valor para ese pixel. En el caso del reconocimiento de imágenes con redes neuronales se puede tener mayor claridad sobre su utilización. Veamos por ejemplo el caso en que se quiere hacer el reconocimiento de una cara. Una cara tiene una serie de características interesantes: los ojos no quedan muy distantes entre sí, bajo los ojos está centrada la nariz y bajo la nariz esta la boca. Estos elementos que conforman la cara se consideran detectores locales, figura 5.20. Los detectores locales buscan en la imagen puntos localmente interesantes.

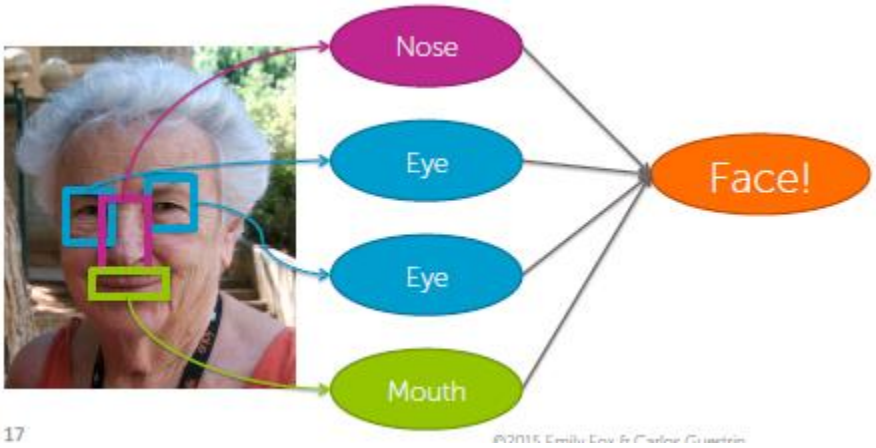


Figura 5.20. Imagen con detectores locales-

Las colecciones de puntos localmente interesantes se combinan para construir clasificadores. Con Dee Learning se aprenden implícitamente estos rasgos (features).

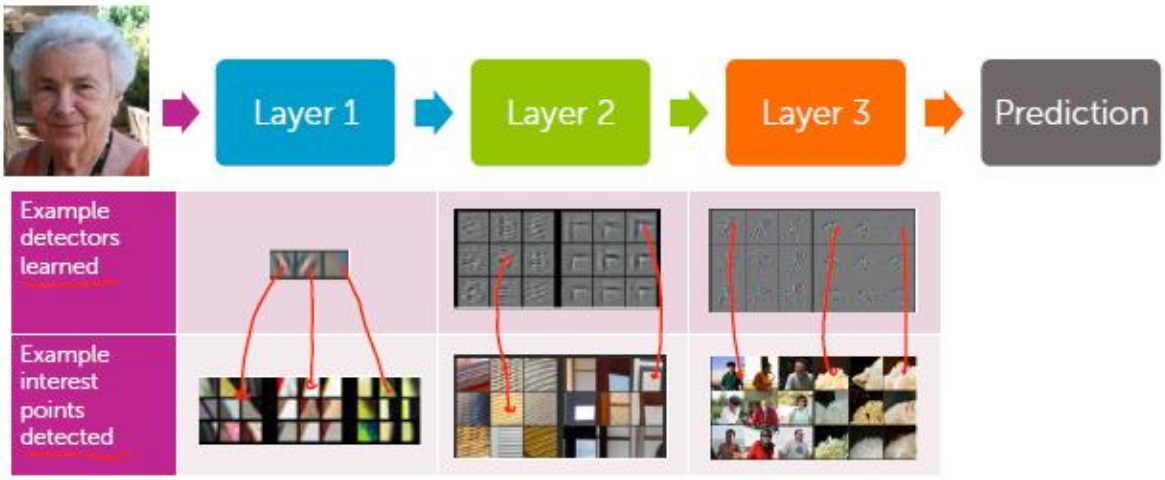


Figura 5.21. En cada capa se va aprendiendo una serie de rasgos, hasta que en la última capa se hace la predicción o reconocimiento.

Pero además se puede pensar que estos rasgos son comunes a una gran cantidad de imágenes. Luego las primeras capas lo que hacen es encontrar rasgos que son comunes a muchas imágenes. Esta es la idea de la trasferencia de aprendizaje en redes con muchas capas.

En el caso de la figura 5.22, la red neuronal se conforma de 8 capas para realizar la clasificación entre perros y gatos. Una vez que la red a aprendido esta clasificación, es decir, se tiene los pesos calculados en cada capa para realizar esta clasificación, se puede tomar la misma red neuronal dejando estos pesos ya calculados hasta la capa 7, incluir otra capa para que prenda a reconocer otra clase de imágenes, como sillas, elefantes, carros y cámaras de fotografía. Realmente es muy provechoso aprovechar el conocimiento de las capas previas.
Figura 5.21.

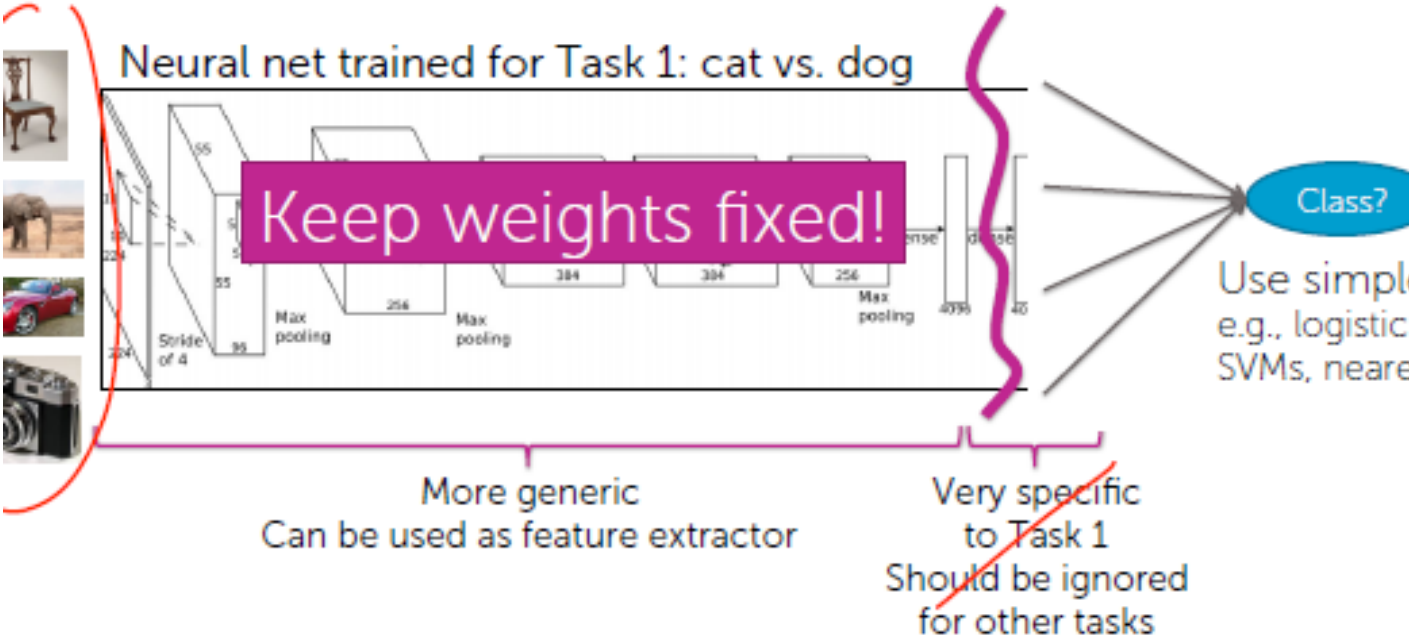


Figura 5.22. Red neuronal de ocho capas. El aprendizaje para clasificar perros y gatos se puede utilizar en otro tipo de reconocimiento o clasificación.

Veamos un ejemplo, para detectar bordes. La idea es que a la matriz que representa una imagen se le aplican ciertos filtros que permiten detectar los bordes. En este caso tenemos la matriz:

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Veamos ahora en que consiste la operación convolución. Tomamos una matriz 3 X 3 y la usamos como un filtro

$$\begin{pmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{pmatrix} * \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$$

Imagen 6 X 6 Filtro 3 X 3

Para aplicar el filtro, primero la matriz de la imagen 4 X 4, se divide en 16 matrices, 3 X 3

$$\begin{pmatrix} 3 & 0 & 1 \\ 1 & 5 & 8 \\ 2 & 7 & 2 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 2 \\ 5 & 8 & 9 \\ 7 & 2 & 5 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 7 \\ 8 & 9 & 3 \\ 2 & 5 & 1 \end{pmatrix} \quad \begin{pmatrix} 2 & 7 & 4 \\ 9 & 3 & 1 \\ 5 & 1 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 5 & 8 \\ 2 & 7 & 2 \\ 0 & 1 & 3 \end{pmatrix} \quad \begin{pmatrix} 5 & 8 & 9 \\ 7 & 2 & 5 \\ 1 & 3 & 1 \end{pmatrix} \quad \begin{pmatrix} 8 & 9 & 3 \\ 2 & 5 & 1 \\ 3 & 1 & 7 \end{pmatrix} \quad \begin{pmatrix} 9 & 3 & 1 \\ 5 & 1 & 3 \\ 1 & 7 & 8 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 7 & 2 \\ 0 & 1 & 3 \\ 4 & 2 & 1 \end{pmatrix} \quad \begin{pmatrix} 7 & 2 & 5 \\ 1 & 3 & 1 \\ 2 & 1 & 6 \end{pmatrix} \quad \begin{pmatrix} 2 & 5 & 1 \\ 3 & 1 & 7 \\ 1 & 6 & 2 \end{pmatrix} \quad \begin{pmatrix} 5 & 1 & 3 \\ 1 & 7 & 8 \\ 6 & 2 & 8 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 3 \\ 4 & 2 & 1 \\ 2 & 4 & 5 \end{pmatrix} \quad \begin{pmatrix} 1 & 3 & 1 \\ 2 & 1 & 6 \\ 4 & 5 & 2 \end{pmatrix} \quad \begin{pmatrix} 3 & 1 & 7 \\ 1 & 6 & 2 \\ 5 & 2 & 3 \end{pmatrix} \quad \begin{pmatrix} 1 & 7 & 8 \\ 6 & 2 & 8 \\ 2 & 3 & 9 \end{pmatrix}$$

Ahora, cada elemento de la nueva matriz se calcula tomando cada una de las submatrices y multiplicándola, de una manera especial, por el filtro.

$$\begin{bmatrix} 3 & 0 & 1 \\ 1 & 5 & 8 \\ 2 & 7 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Así se calcula el primer elemento de la nueva matriz:

3 X 1 + 1 X 1 + 2 X 1 + 0 X 0 +5 X 0 + 7 X 0 + 1 X -1 + 8 X -1 + 2 X – 1 = - 5

Se hace el mismo cálculo para todas las otras submatrices, obteniéndose la siguiente matriz:

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

De esta manera al hacer la convolución de una matriz 6 X 6 con un filtro 3 X 3, obtenemos una matriz 4 X 4. De la misma manera se calculan los siguientes elementos de la matriz 4 X 4:

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

Algunos de los problemas de visión computacional que se resuelven con CNN son: clasificación de imágenes, detección de objetos y transferencia de estilo neuronal. Un problema importante con los problemas de visión computacional es que los datos de entrada son realmente cuantiosos.

5.8.2 Polling

Pooling

Pooling es el proceso de eliminar pixels en una imagen, mientras se mantiene la semántica del contenido dentro de la imagen. Esto se explica mejor visualmente. La figura 5.23, muestra el concepto de un m*ax* pooling.

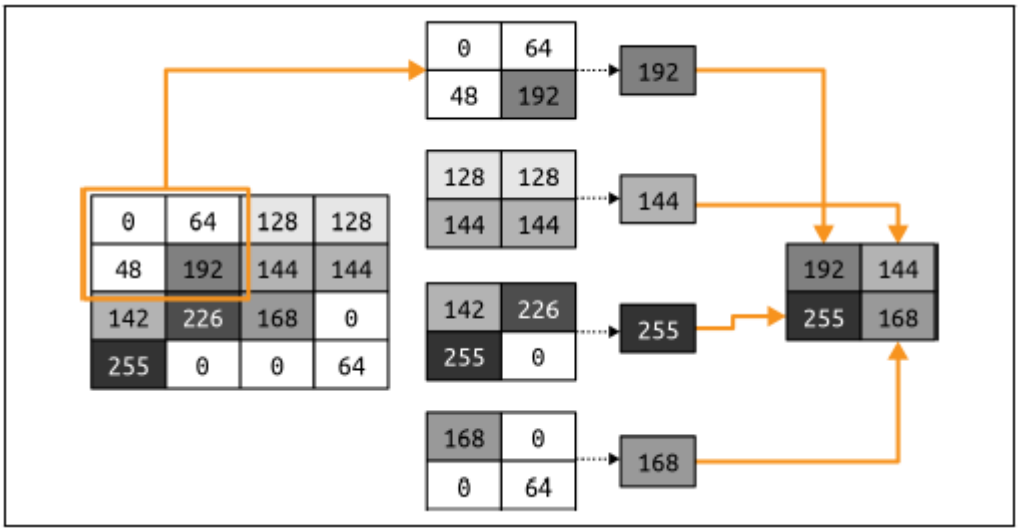


Figura 5.23. Demostración de max pooling
En este caso, considere que la caja de la izquierda son los pixeles que están una imagen monocromática. Los agrupamos en arreglos de 2 X 2, en este caso los 16 bits se agrupan en 4 arreglos de 2 X 2. Estos se llaman *pools*.

Cuando seleccionamos el máximo valor, en cada uno de los grupos, y los reagrupamos en una nueva imagen. De esta manera los pixels de la izquierda se reducen en un 75% (de 16 a 4), con el valor máximo de cada pool conformando la nueva imagen. Ascendente de la figura 5.24. Con las líneas verticales ampliadas, después de que se aplica el max pooling.

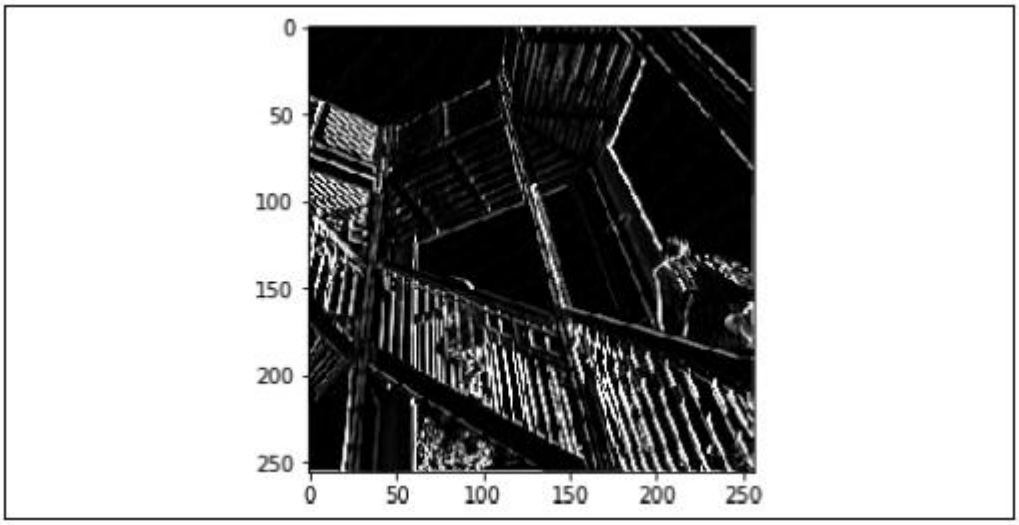
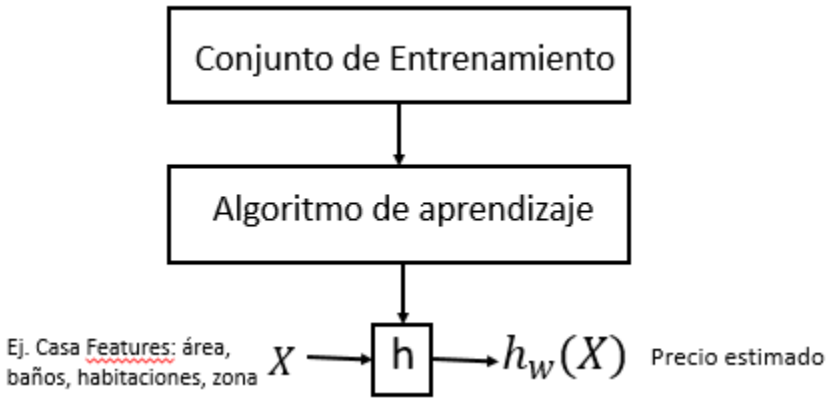


Figura 5.24. Ascendente después del filtro vertical y el max pooling

Note que las características filtradas no solo se han mantenido, sino que además han mejorado. También el tamaño de la imagen ha cambiado de 512 X 512 a 256 X 256, un cuarto del tamaño original.

Los datos en el aprendizaje supervisado, incluye la respuesta correcta Y_r o valores rotulados, para cada ejemplo X, el par (X, Y_r).

Notación
m = número de ejemplos de entrenamiento.
X = variable de entrada, es un vector con todas las características (features) del ejemplo.
n = número de características (features) de la variable de entrada.
Y = variable de salida, valores objetivo (target), o valores rotulados.



En las diferentes heurísticas de ML, se define una función de costo J que es la diferencia entre la hipótesis que produce un modelo $h_w(X)$ y el valor real Y en el conjunto de aprendizaje:

$$J(W) = \frac{1}{2m}\left(\sum_{i=1}^{m} (h_w(X) - Y)^2\right)$$

Donde $\frac{1}{2}(h_w(X) - Y)^2$, es el error cuadrático que se ha trabajado, para el conjunto de aprendizaje. Cuando se tiene en el conjunto de aprendizaje m ejemplos, lo que nos dice la ecuación es que se toma el promedio de todos los errores al entrenar la red con todos los m ejemplos del conjunto de entrenamiento.

La función de costo también se conoce como función de pérdida y tal vez es el nombre que más se utiliza. Para ML hay varias funciones de pérdida.

5.9. Ejercicios.

1. Con base en el programa Clase_NNA3, desarrolle dos redes neuronales con 2 capas oculta, una que aprenda a reconocer una compuerta NAND y la otra aprenda a reconocer una compuerta XOR.
2. Con base en la librería tensorflow, descargue el data set fashion MNIST. Haga una clasificación de prendas de vestir. Explique cada una de las funciones y las principales instrucciones, saque conclusiones.
3. Consiga un data set de cualquier tipo, puede ser de https://www.kaggle.com/datasets, estudie sus características (features) y su rótulo. Diseñe una red neuronal y haga ejemplos con base en los pesos aprendidos.

5.10 Bibliografía

1. K. He, X. Zhang, S. Ren y J. Sun, Deep Residual Learning for Image Recognition, Microsoft Reasearch.

3. Laurence Moroney, AI and Machine Learning for Coders *A Programmer's Guide to Artificial Intelligence.*

4. A. Ng, Machine Learning, Coursera

6. INTRODUCCION A MACHINE LEARNING

José J. Martínez P.
Jmcursosingenieria111@gmail.com
Marzo 2026

6.1 Introducción

Machine Learning, ML, es un término que ya se venía utilizando desde el siglo pasado. Aunque, su florecimiento no es muy viejo, solo se logró hasta 2005, cuando G. Hinton, uno de los ganadores del Premio Nobel de Física 2024, publicó un artículo en el que mostraba cómo al entrenar una red neuronal profunda, se podían reconocer dígitos escritos a mano, con una precisión del 98%. Como se ha mencionado, la historia de las redes neuronales ha sido una historia de amores y desamores y a finales de los 90s se había abandonado su investigación en estas redes. Con este artículo se demostró que con Deep Learning era posible lograr cosas que ninguna otra técnica de ML había hecho. El entusiasmo se extendió a muchas otras áreas de aplicación y digamos que ahora estamos viviendo el auge de ML, que, considero, va a perdurar bastante.

Sin embargo, hay otras técnicas básicas diferentes a las RNAs, para enfrentar los mismos y otros problemas, especialmente Bayes simple, K-NN Nearest, SVM (Support Vector Machine) y árboles de decisión, que consideramos son importantes de mencionar dentro de la gama de herramientas de IA. Estas últimas las presentamos en un apéndice.

Supongamos el siguiente problema: traducir un texto de un idioma a otro y solucionarlo con las técnicas actuales de programación. Para hacer un programa de computador que realice esta tarea. Inicialmente, hay que estudiar la gramática de ambos lenguajes, con el fin de codificar sus reglas; algo que no es tan simple. Luego incluir los diccionarios de ambos lenguajes, incluir también la codificación de las búsquedas y seleccionar las palabras más apropiadas en el lenguaje de destino; otra tarea de programación complicada.

Pero hay más, ¿cómo trabajar con palabras homógrafas u homónimas? Por ejemplo, la palabra "can" en inglés, en español, puede ser el verbo poder o puede ser simplemente una lata. ***Todo depende del contexto,*** una frase con la que nos volveremos a encontrar. ¿Pero cómo codificamos apropiadamente ese contexto? La solución de este tipo de problemas, bajo este concepto de programación, es extremadamente difícil y por esto los traductores de lenguajes, antes de ML eran muy pobres.

Ahora las cosas han cambiado. Se encontró que la solución de muchos problemas complejos se encuentra en los datos. Utilizando una gran cantidad de datos provenientes de diferentes fuentes de la web, Google tiene un modelo de traductor pre-entrenado que va mejorando sus resultados, de manera tan sorprendente, que en algunos casos supera el nivel de un traductor humano.

Con base en este punto de vista, ahora se han desarrollado cientos de aplicaciones, que silenciosamente le dan poder a cientos de productos que usamos regularmente; desde mejores

recomendaciones hasta búsquedas por voz. Es por esto que, nos pueden surgir algunas preguntas:

- ¿Qué es ML?
- ¿Por qué el uso de ML?
- ¿Para qué se usa ML?
- ¿Cómo aprende ML?

6.2 ¿Qué es ML?

Veamos varias definiciones de ML:

- "ML es el campo de estudio que le da a los computadores la habilidad para aprender sin que sean explícitamente programados". Arthur Samuel 1959.
- "ML es la ciencia o el arte de programar los computadores, de manera que aprendan a partir de los datos".
- Una simple: "Sistemas que mejoran su comportamiento con respecto a una tarea con más y más experiencia o datos"
- Más orientada a la ingeniería: "Se dice que un programa de computador aprende de la experiencia **E**, con respecto a una tarea **T**, con una medida de comportamiento **P**, si su comportamiento sobre **T**, medido por **P**, mejora con la experiencia". Tom Mitchel 1998.

Otra forma de ver ML. En la figura 6.1, se puede observar, ML en comparación con la programación tradicional tomando como ejemplo encontrar la raíz cuadrada de un número.

En el caso de la programación tradicional podemos programar el método de Newton y hallar la solución. En el caso de ML, tendríamos muchos ejemplos con resultados correctos de la raíz cuadrada y a través de regresión podríamos encontrar el modelo que me permite calcular cualquier raíz cuadrada, y para un caso concreto encontrar la solución.

Figura 6.1. Programación tradicional versus ML.

Para mayor claridad, vamos a trabajar un ejemplo que es el filtro que tienen los sistemas de correo electrónico para detectar correos engañosos, que se conocen como SPAM. El filtro spam que usa nuestro correo, es un programa de ML, que, con base en instancias de emails rotuladas como spam, observa qué mensajes del correo marca o no como spam; y con qué instancias de correos normales, aprende cómo filtrar mejor el spam. Esas instancias de emails que usa el sistema para aprender se llaman el conjunto de entrenamiento, o dataset.

Volviendo a la última definición de ML, la tarea **T** del filtro Spam es rotular los correos que llegan como spam, correo no deseado, cuando esos correos presentan ciertos atributos que el filtro Spam tiene clasificados como comunes a todos los correos spam. La experiencia **E**, es el conjunto de correos buenos y malos, el data set, que tiene el sistema y del cual aprende. Este data set, lo ha ido recabando el sistema de la interacción permanente con sus usuarios en todo el mundo. Mientras que **P** es la predicción que hace el sistema de si un determinado correo es spam o no; también se conoce como el comportamiento del sistema, que depende de qué tan bien hace la predicción y se le asigna una precisión. Para esto se usa la Matriz de Confusión, figura 6.2, que se usa mucho en problemas de clasificación

Matriz de Confusión		Predice	
		Positivo	Negativo
Real	Positivo	VP	FN
	Negativo	FP	VN

VP: Verdadero Positivo FN: Falso Negativo

FP: Falso Positivo VP: Verdadero Negativo

$$Precisión = \frac{VP + VN}{VP+VN+FP+FN}$$

Figura 6.2. Matriz de Confusión, permite encontrar la precisión de un sistema de clasificación por ML.

6.3 Porque el uso de ML

Volvamos a nuestro ejemplo del filtro Spam. Digamos que vamos a escribir el sistema clasificador, el filtro spam, usando técnicas de programación tradicional. Siguiendo unos pasos básicos de Ingeniería de Software, primero debemos enterarnos cómo son los correos no deseados, cómo es la forma de los encabezados, la del tema y la del contenido. Aparecen palabras como “gratis”, “regalo”, “tarjeta de crédito”, etcétera.

Con base en toda esta información se encuentran estas palabras y algunos patrones que casi siempre se presentan en los correos spam. De manera que como la solución tradicional se entra a programar el filtro; para que cada una de estas palabras y patrones, permitan que eli programa detecte los correos no deseados.

Con un gran esfuerzo de programación, puede ser que se llegue a que ese filtro Spam funcione correctamente. Pero solo lo va a hacer durante un determinado tiempo, pues los que producen los correos Spam se dan cuenta y van a buscar otras palabras y otros patrones, para burlar el filtro programado. Ahora, el filtro que se programó deja de funcionar un tiempo, hasta cuando, los desarrolladores del filtro programado se dén cuenta de que su sistema está dejando pasar correos spam.

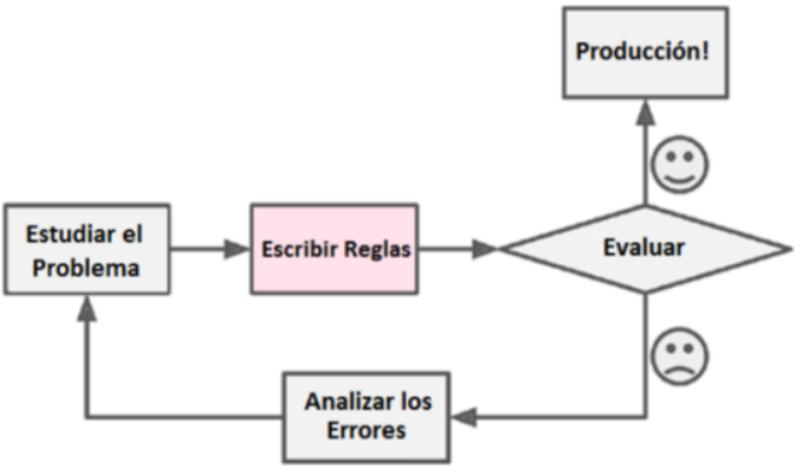


Figura 6.3 Enfoque de programación tradicional

Ahora las nuevas palabras pueden ser “pandemia”, “soledad”, “ayuda”, “anticovid”. Entonces, después de haber logrado esta tarea, con su correspondiente gasto de tiempo, en el programa se deben incluir estas nuevas palabras y encabezados; otro gran esfuerzo. Pero nuevamente quienes producen los spam buscan nuevas formas de que sus correos sean aceptados como correos buenos. De manera que después de un análisis, se tendrá que volver a actualizar el programa del filtro; y así sucesivamente. Entonces llegamos a la conclusión de que definitivamente esta no es la mejor manera de mantener un filtro actualizado y funcionando permanentemente.

En contraste, un filtro de spam basado en técnicas ML, aprende automáticamente qué palabras y frases son buenos predictores de spam, detectando patrones inusualmente frecuentes de las palabras en ejemplos spam, comparados con los ejemplos normales. El programa es mucho más corto, más fácil de mantener y probablemente más preciso. Por lo que un filtro spam basado en ML se dará cuenta automáticamente que “pandemia”, “soledad”, “ayuda”, “anticovid”. se han vuelto inusualmente frecuentes en los emails rotulados por los usuarios como spam y comenzará a marcarlos sin necesidad de reprogramarlo.

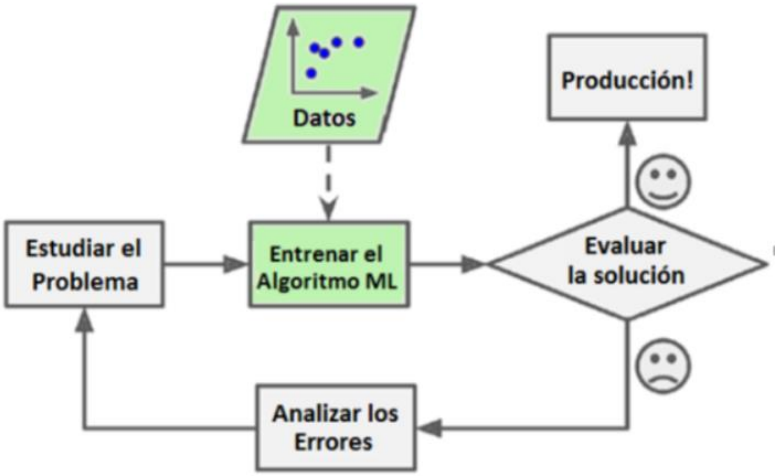


Figura 6.4. Enfoque de programación con ML.

En la figura 6.4, se puede observar cómo funciona la programación ML. El primer paso, lo mismo que en la programación tradicional se estudia el problema, pero luego se cambia, se consigue el data set de spams y hams y se entrena un algoritmo de ML, que es realmente un ***modelo***. Se evalúa su solución hasta cuando el modelo funcione ben y se lanza a producción. Sin embargo, en la medida en que los spams van cambiando, los usuarios van rotulando qué correos son no deseados, el modelo va aprendiendo y se va ajustando permanentemente a las nuevas formas de spam. Así, la actualización del filtro es automática.

Aplicar técnicas de ML para explorar en grandes cantidades de datos, puede ayudar a descubrir patrones que no eran evidentes, lo que se conoce como "*minería de datos*". Otra área donde sobresale ML es en problemas que o son muy complejos para el enfoque tradicional, o simplemente hasta ahora no hay un algoritmo conocido.

En resumen, ML es ideal para:

- Solucionar problemas para los cuales existen soluciones que requieren mucho refinamiento o largas listas de reglas: un algoritmo ML generalmente puede simplificar el código y comportarse mejor que el enfoque tradicional de programación.
- Solucionar problemas complejos: para los cuales se usan fundamentos de enfoques tradicionales que no son una buena solución.
- Ambientes fluctuantes: un sistema ML puede adaptarse a nuevos datos.
- Sacar ideas sobre problemas complejos con gran cantidad de datos.

6.4 Aplicaciones de ML

Son muchas las áreas de aplicación de ML, que dejan abierta a muchos nuevos desarrollos en diferentes campos, tanto para la predicción o regresión, como para clasificación:

1. **Traductores automáticos de lenguajes.** Son muy utilizados en diferentes tipos de aplicaciones, pues son universales ya que permiten la traducción de un texto en un lenguaje a otro de más de 50 posibles lenguajes con una muy alta precisión. Esta es una aplicación de ML, muy importante porque permite con facilidad la comunicación entre los seres humanos.
2. **Filtros spam para programas malignos** (malware). Como lo vimos en nuestro ejemplo, la llegada de correos mal intencionados, pueden inducir a generarle problemas al usuario, lo mismo puede suceder con cualquier otro tipo de información textual, como puede ser la publicidad engañosa. Además, poder filtrar los programas engañosos o virus, es de mucha utilidad para el usuario normal.
3. **Reconocimiento de voz.** Incluye paso de voz a texto, en robótica para comandos por voz, identificación de voz, análisis de sentimientos, síntesis de voz y otras. Es un componente clave para muchas aplicaciones.
4. **Reconocimiento de imágenes**. Es una de las aplicaciones más comunes de ML, por la cantidad de utilidad que tiene en campos que van desde la robótica, la seguridad hasta los sistemas de recomendación por imágenes.

5. **Recomendación de productos.** O, Sistemas de Recomendación, se usan mucho en comercio electrónico como en Amazon, AliExpress, Netflix. Una vez uno ha seleccionado un producto las redes sociales lo invaden a uno de información referente al producto.
6. **Asistente personal virtual.** Sistemas que ayudan a la toma de decisiones gerenciales leyendo información directamente de las bases de datos empresariales, haciendo promedios y resúmenes para adecuarlas al ambiente. También en herramientas de negociación.
7. **Vehículos autónomos.** Los carros autónomos ya están en prueba en varios lugares del mundo, lo mismo que muchos robots móviles en empresas industriales. También en aviónica, drones, y para exploración espacial, roberts que en determinadas circunstancias funcionan de manera autónoma.
8. **Detección en línea de fraudes**. Las transacciones en línea son permanentes y se utilizan en todo el mundo, aquí la utilización de ML hace estas transacciones confiables y seguras.
9. **Predicción del tráfico.** Las vías del mundo están cada vez más congestionadas y los costos en tiempo y dinero de tener personas y mercancías varadas, es muy alto. De manera que cuando se quiere predecir una ruta hay sistemas confiables que lo hacen como "wase", que funciona en tiempo real, con base en tecnologías y empresas que proveen estos servicios satelitales.
10. **Diagnóstico médico.** ML se comienza a utilizar en diagnóstico médico, especialmente apoyados en tomografías. Ayuda en diagnóstico y hallazgo de tumores de seno y cerebrales. Aquí es importante destacar la riqueza que hay en las historias clínicas, que se tiene de todos los usuarios de los servicios médicos, de cuya riqueza todavía no se tiene conciencia. Es posible que ya haya equipos para este tipo de diagnóstico que directamente incluyan hardware especializado con Deep Learning, para diagnóstico rápido.
11. **Negocios en mercado de valores.** El uso de ML en mercado de valores esta muy extendido. Aquí la variable tiempo es fundamental, en algunos negocios de Fintech.

La lista anterior viene ordenada por la utilización de datos independientes del ambiente hasta de datos muy dependientes del ambiente. Lo que nos lleva a tratar de definir ¿qué es aprendizaje?

Aprendizaje.

Según Wikipedia, es el proceso por el cual se adquieren y desarrollan *habilidades, conocimiento*, *conductas* y *valores*.

Habilidades son un conjunto de capacidades, destrezas y talentos de un humano o un animal para realizar ciertas acciones.

Conocimiento, hace referencia a la comprensión teórica o práctica de un tema; se adquiere a través del estudio o la experiencia. Puede ser implícito como las habilidades de un humano o un animal; o explicito como el conocimiento humano escrito. El paso del conocimiento humano de implícito a explicito es complejo y no logra reflejarse completamente.

Conducta, o comportamiento, es el conjunto de reglas que utilizan los seres vivos en su entorno ante la presencia o ausencia de estímulos.

Valores, son cualidades de un objeto, son agregados a las características físicas del objeto.

Entonces el aprendizaje en ML se basa en el principio de que el dataset, el gran conjunto de datos, guarda conocimiento alrededor de un tema específico, en otras palabras, guarda conocimiento que a través de sus técnicas se vuelve conocimiento tácito. Pero es posible que ese dataset no incluya todo el conocimiento necesario para que el modelo que se construya con él no sea el más apropiado para hacer predicciones o clasificaciones.

6.5 Algoritmos de ML

Hay muchos otros algoritmos de ML, diferentes a las redes neuronales, que se utilizan, dependiendo de la aplicación, con bastante frecuencia. Entre ellos están:

- Regresión lineal
- Regresión logística
- Algoritmo Bayes simple
- Algoritmo Vecinos más cercanos K-nearest o KNN
- Algoritmo SVM (Support Vector Machine)
- Algoritmo Arboles de decisión

Estos algoritmos se presentan en el Apéndice de este capítulo.

6.6 Tipos de aprendizaje en ML

Ya hemos hablado de aprendizaje cuando estudiamos el aprendizaje supervisado para el algoritmo de retropropagación en RNAs. Además, por el tipo de aprendizaje se permite de cierta manera una clasificación de algoritmos de ML. Hay cuatro categorías importantes de aprendizaje: aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semi-supervisado y aprendizaje reforzado RL.

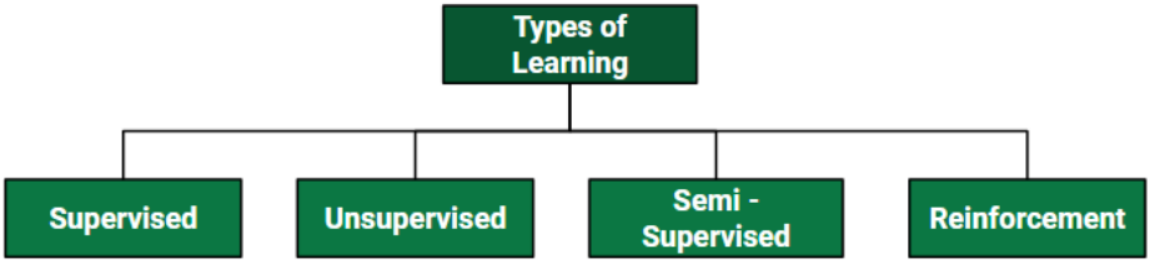


Figura 6.5. Tipos de aprendizaje.

6.6.1 Aprendizaje supervisado

Como ya se ha visto, en el aprendizaje supervisado el conjunto de entrenamiento con el que se alimenta el algoritmo incluye las soluciones deseadas llamadas rótulos o labels.

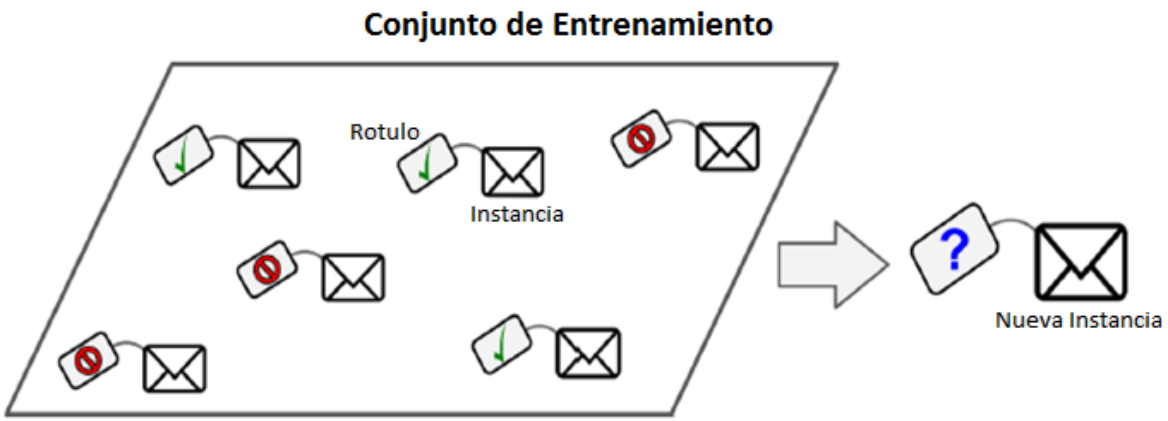


Figura 6.6. Un conjunto de entrenamiento rotulado para clasificación de spam.

El modelo se ajusta a los datos de entrenamiento que comprenden las entradas y las salidas. Las técnicas de aprendizaje supervisado se utilizan en:

1. Modelos de clasificación. Reconoce y agrupa ideas, objetos en categorías predefinidas. Por ejemplo:
   Clasificación de correos Spam y Ham
   Retención de clientes
   Clasificación de imágenes
   Reconocimiento de caras
   Salud
2. Regresión. Se da cuando se crea una ecuación, usando diferente valores de entrada con sus pesos específicos determinados por el valor total del impacto en el resultado y los usamos para generar un estimado del valor de salida.
   Por ejemplo, las aerolíneas se basan fuertemente en ML para predecir que tanto deberían cobrar por el boleto de avión. Usan varias características, como los días que faltan para el vuelo, el día de la semana, el destino; para usarlos con el fin de predecir el costo del boleto de un vuelo determinado que maximice sus ganancias.
   Otra tarea es predecir un valor numérico objetivo, como el precio de un carro. Sí se tiene un conjunto de datos con características como: kilometraje, edad, marca, y otros, llamados predictores, se puede hacer un modelo de regresión para entrenar el sistema. En la figura 6.7, se tiene un modelo de regresión ajustado con el conjunto de entrenamiento, en este caso con solo un atributo (feature), el kilometraje, para predecir el valor de un carro. En general hay varias características de entrada.

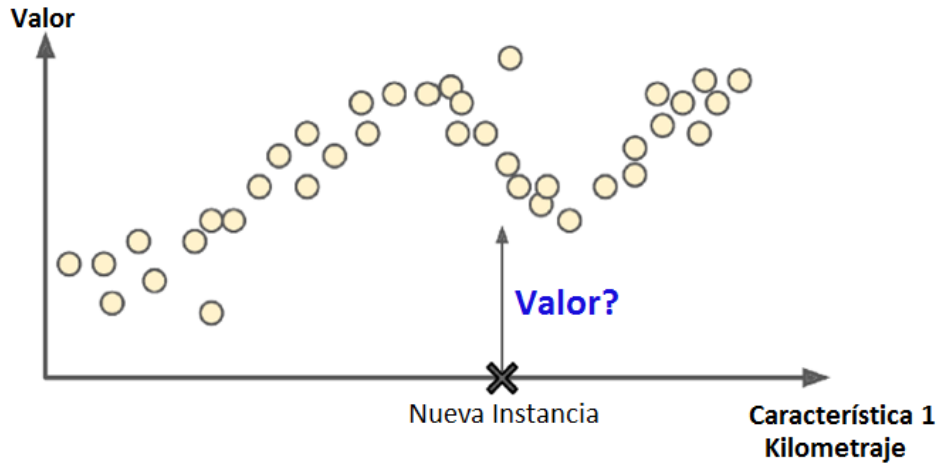


Figura 6.7. Un conjunto de entrenamiento rotulado para regresión.

6.6.2 Aprendizaje no supervisado

Como se puede intuir, el aprendizaje no supervisado se realiza con un dataset que no tiene rótulos. Se utiliza cuando se usan algoritmos de ML para analizar y agrupar con base en datasets no rotulados, sin la necesidad de la intervención humana, por ejemplo en:

1. Agrupamiento (clustering), por ejemplo cuando las organizaciones tratan de hacer segmentación de clientes, para hacer mercadeo efectivo. Es realmente crítico que comprendan quienes son sus clientes para que puedan conectar con ellos de la manera más productiva.

   En la figura 6.8, se puede apreciar un conjunto de datos sin rótulo.

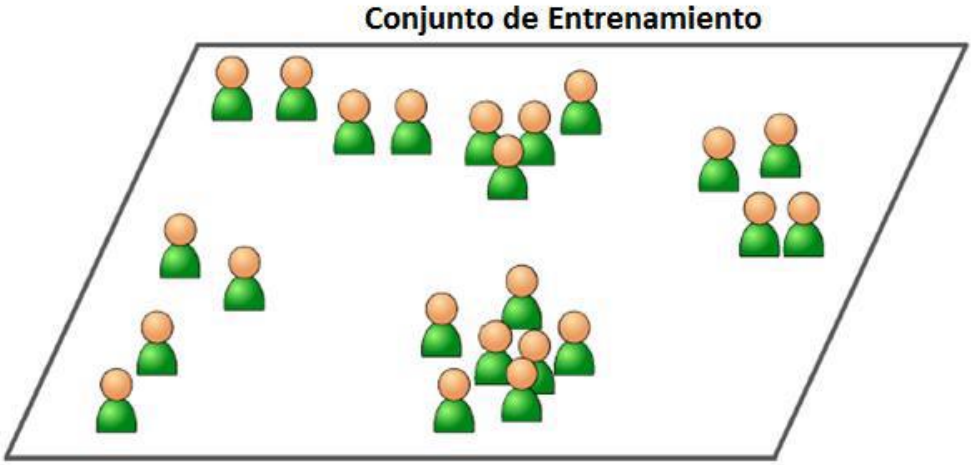


Figura 6.8. Un conjunto de entrenamiento rotulado para aprendizaje no supervisado.

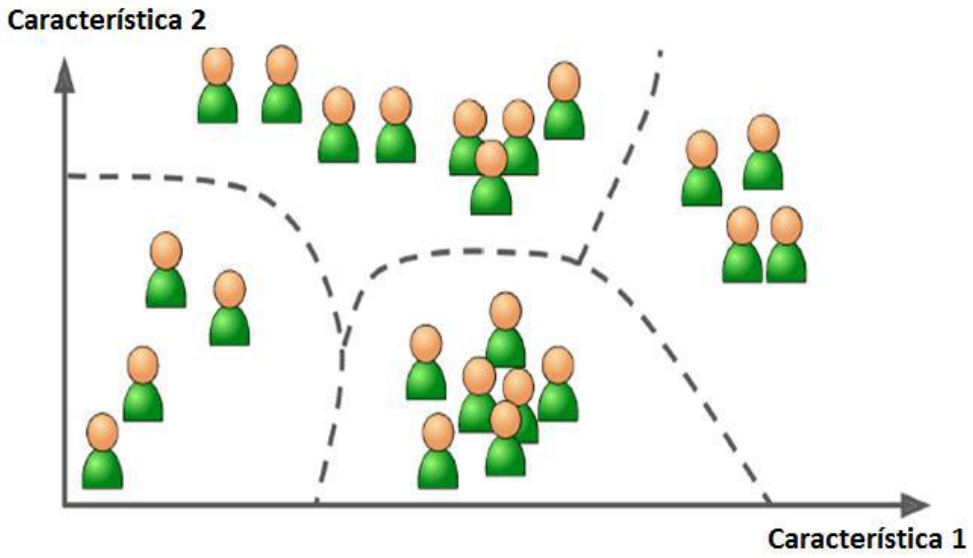


Figura 6.8. Agrupamiento, resultado de entrenamiento con aprendizaje no supervisado.

2. Otro ejemplo de tarea no supervisada es la detección de anomalías, por ejemplo, detectar transacciones inusuales en tarjetas de crédito para prevenir el fraude; encontrar defectos de fábrica o eliminar valores atípicos en los datasets para alimentar otro algoritmo de ML. En la figura 6.9, se puede observar cómo al llegar una nueva instancia de una transacción comercial, el algoritmo puede detectar si la transacción es válida o se trata de una anomalía, es decir de un fraude.

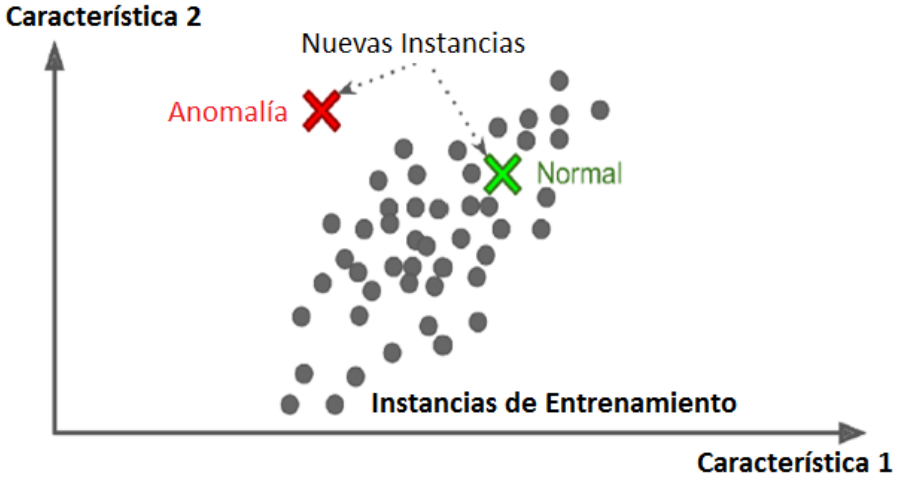


Figura 6.9. Detección de anomalías.

3. Reducción de la dimensionalidad. Se refiere a técnicas que reducen el número de variables de entrada, cuando en el número de variables de entrada en un dataset es muy alto, es posible que haya algunos parámetros redundantes.
   Los algoritmos de visualización son buenos ejemplos de aprendizaje no supervisado. Al alimentarlos con datos complejos y sus rótulos, estos algoritmos obtienen una representación 2D o 3D de sus datos, que se puede plotear fácilmente. Y tratan de

preservar la estructura y la manera en que se puede comprender cómo están orientados y organizados los datos y quizá encontrar patrones no sospechados.

6.6.3 Aprendizaje semi-supervisado

El aprendizaje semi-supervisado (SSL) es una técnica de ML, que se presenta cuando hay una pequeña parte de datos rotulados y cantidades de datos no rotulados, para entrenar un modelo predictivo. Su importancia se debe a lo costoso de la rotulación.

Algunos servicios de almacenamiento de fotos, como Google fotos, son buenos ejemplos de este tipo de aprendizaje. Una vez usted descarga las fotos de su familia allí, este reconoce automáticamente la persona que aparece en diferentes fotografías.

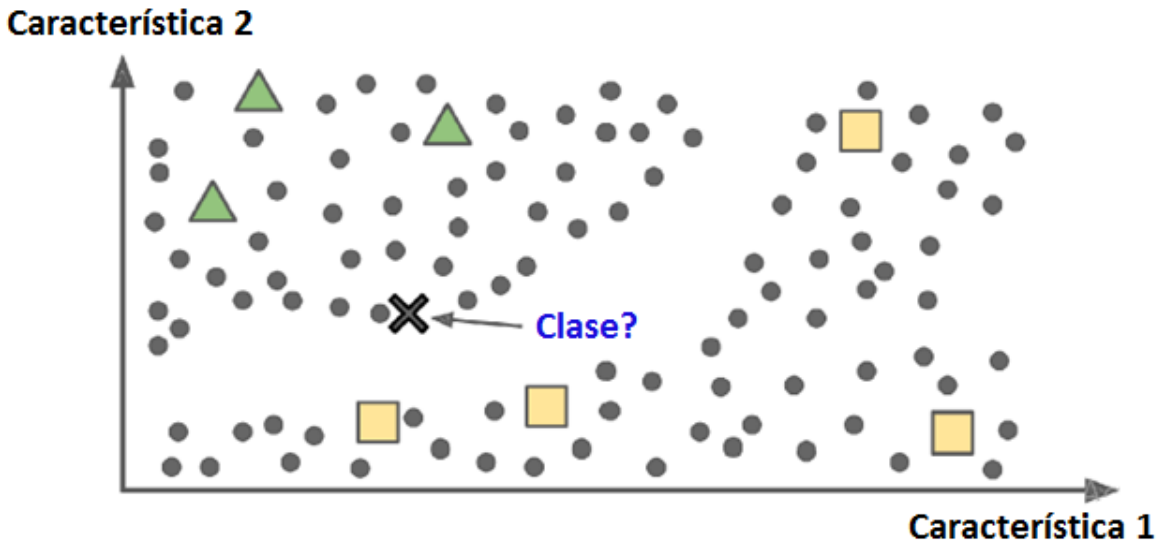


Figura 6.10. Aprendizaje semi-supervisado.

También son varios los proyectos de este tipo que han desarrollado varias redes sociales como FaceBook. Un algoritmo bastante popular es representar los datos rotulados y no rotulados en forma de grafo y luego aplicar un algoritmo de propagación de rótulos. Este difunde rotulaciones hechas por un humano a través de toda la red del data set.

6.6.4 Aprendizaje reforzado

En este tipo de aprendizaje, normalmente se tiene un “agente”, es el sistema de aprendizaje, que toma acciones en un ambiente definido. Aquí se entrenan los agentes para seguir una política, es decir efectuar una secuencia de decisiones.

Para el entrenamiento, el agente observa el ambiente, toma decisiones y ejecuta acciones. El agente recibe una retroalimentación en términos de recompensas de retorno dependiendo de si la acción fue exitosa o penalidades, en caso contrario. Debe aprender por sí mismo cuál es la

mejor estrategia, la política, para obtener la mayor recompensa. En el tiempo, una política define qué acción debe escoger el agente cuando está en una situación dada.

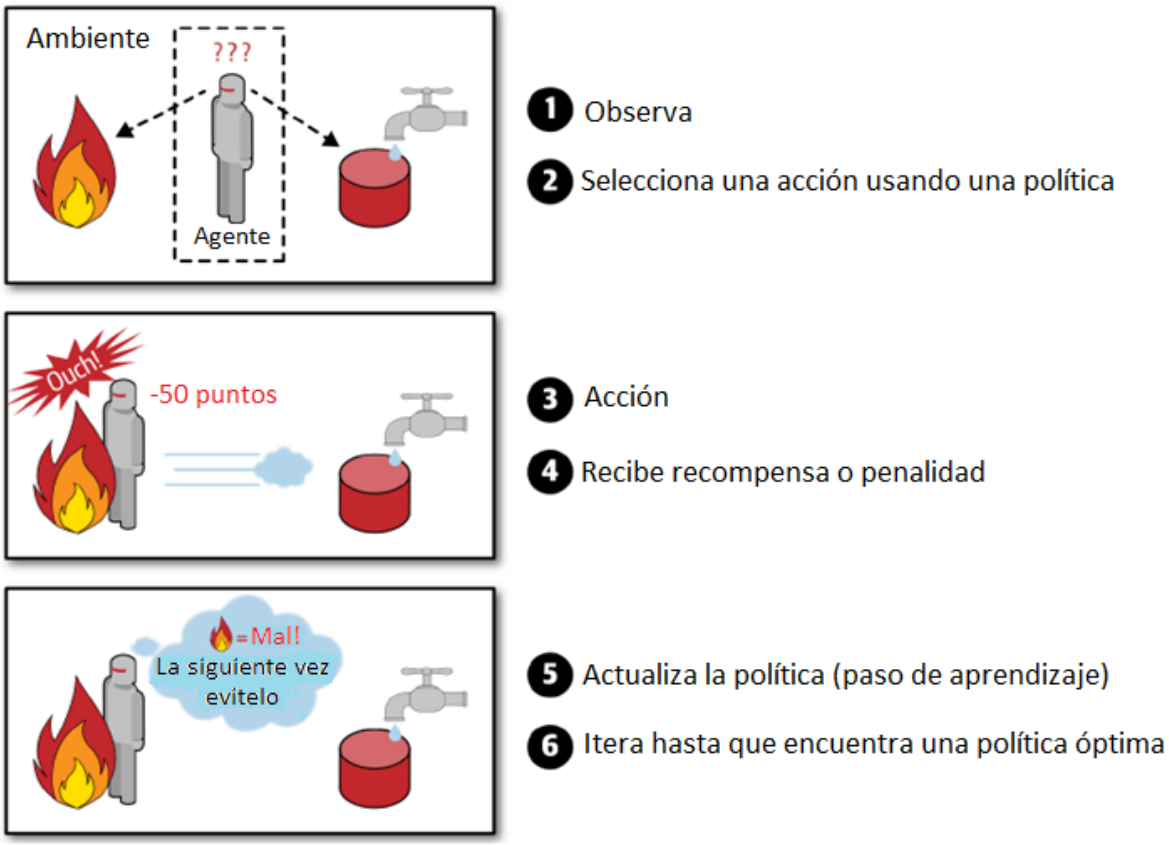


Figura 6.11. Aprendizaje reforzado.

Al principio, el agente no conoce nada sobre el ambiente, por lo que toma las acciones de forma aleatoria. Si una acción trae una recompensa positiva, el agente deberá aprender a escoger esa acción más frecuentemente, mientras que, si una acción lleva a una penalidad, el agente tenderá a escoger esa acción menos frecuentemente. Así, **el agente aprenderá a escoger las acciones que maximicen la suma de recompensas recibidas,** que se conoce como **retorno.**

Un muy buen ejemplo del mundo real son los carros autónomos, que tiene varios factores: límites de velocidad, colisiones, zonas de conducción y otros.

Hay tres tipos de algoritmos de aprendizaje por refuerzo:

- **Q-Learning**. El más importante, calcula el refuerzo ´por estados y acciones. Se usa cuando hay un número finito de estados y acciones.
- **Iteración de política**. Calcula el refuerzo por estados y acciones siguiendo dos pasos: evalúa la política en un paso, seguido por un paso de mejora en la política. Hay un agente y un dominio de estados y acciones. La tarea es encontrar una política que cause el refuerzo para incrementar cada estado inicial, sin causar el refuerzo de ningún otro estado sucesor para disminuirlo. Se usa cuando hay un número infinito de estados y acciones.

- **Iteración de valor**. Calcula el refuerzo para estados y acciones usando una señal de refuerzo determinada por una función de refuerzo. Se usa cuando conocemos una ecuación de ambiente y se necesita encontrar una función acción-valor que es una Q-función que nos da el refuerzo para cada estado y acción. Se debe tener información completa sobre la ecuación de transición del ambiente.

Ver ejecución del juego Bong, en un Notebook de Jupyter.

6.7 Los datos

En el capítulo 1, hablamos de la utilización de los datos como una estrategia para realizar IA, lo que se puede observar directamente en ML. Para resaltar la importancia de los datos, vamos a hablar sobre la irrazonable efectividad de los datos.

En un artículo famoso publicado en 2001, de M. Banko y E. Brill de Microsoft, mostraron que algoritmos muy diferentes a ML, incluyendo los más simples, se comportan casi idénticamente bien, alrededor de la solución de un problema complejo una vez se tengan los datos suficientes: eliminar la ambigüedad en los lenguajes naturales. La idea de que los datos son más importantes que los algoritmos para solucionar problemas complejos fue popularizada por P. Norving, en el artículo "La irrazonable efectividad de los datos".

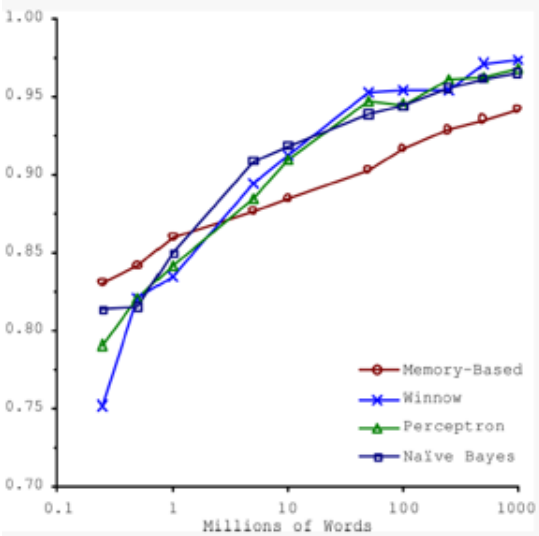


Figura 6.12. La irrazonable efectividad de los datos.

Con la digitalización, han aparecido muchas fuentes de datos, de las cuales se pueden extraer nuevos patrones y conocimiento, que pueden ser útiles para el desarrollo de nuevas aplicaciones, en diferentes dominios. Para crear modelos efectivos y de calidad de ML, se deben seguir una serie de pasos que garanticen la consistencia del dataset, como es la preparación de los datos.

El procesamiento de datos incluye tanto la preparación de datos como la ingeniería de características. Antes de preparar los datos, es necesario comprender el resultado que se requiere del modelo de ML que se va a entrenar y, por lo tanto, definir los atributos de los datos que lo van a configurar. Con el resultado que se busca, los datos que se van recopilar se identifican fácilmente

y, por lo tanto, se definen sus requisitos de calidad y valor. Esta articulación con el problema garantiza que se sigan los pasos correctos en la preparación del dataset.

6.7.1 Preparación de los datos

Los modelos de ML son específicos de su fuente de datos, por lo que la credibilidad de la fuente y la utilidad de los datos recopilados son esenciales. Es posible que un modelo de ML poderoso, pero entrenarlo con datos erróneos, produzca predicciones erróneas.

La preparación de datos minimiza los errores y permite la monitorización de futuros errores. Esto garantizará que el modelo de ML se entrene con los datos correctos y, por lo tanto, que la predicción del modelo resultado sea más preciso. Un análisis de exploratorio de los datos proporciona un resumen del dataset y permite realizar los cambios necesarios en el formato. Además, la preparación de datos ayuda a adaptar los datos a los requisitos del modelo de ML.

6.7.2 Recolección de datos

La recolección de datos es la etapa inicial de la preparación de datos e implica la decisión sobre el dataset en función del resultado esperado del modelo de ML que se va a entrenar. En esencia, la recopilación del conjunto de datos correcto garantiza el resultado correcto. La recopilación de datos consiste en la adquisición, el etiquetado, la ampliación, la integración y la agregación de datos.

6.7.2.1 Adquisición de datos.

Implica identificar la fuente de datos, definir la metodología de recopilación y convertir los datos recopilados en formato digital para su cálculo. La fuente de datos puede ser primaria, es decir, se obtiene directamente de las personas, objetos o procesos estudiados. Cuando la fuente es una parte que ya ha recopilado datos previamente; se denomina fuente secundaria. En esta etapa, se utiliza el análisis exploratorio de datos (EDA), técnica que busca comprender las características y atributos de los dataset. La metodología de recopilación de datos varía según el resultado esperado. Las herramientas y técnicas estadísticas se aplican tanto en la recopilación de datos cualitativos como cuantitativos.

6.7.2.2 Etiquetado de datos

Es el proceso que busca colocarle el rótulo correcto a cada conjunto de características de un data set, para su utilización en aprendizaje supervisado. Cuando faltan esos rótulos, es un proceso necesario, que es bastante costoso pues generalmente incluye trabajo humano. Por ejemplo en agricultura, se debe utilizar para identificación de enfermedades.

6.7.2.3 Aumento de datos

Es una estrategia de preparación de datos que se utiliza para aumentar la diversidad de datos en el entrenamiento de modelos de ML. Implica el desarrollo de nuevos datos de entrenamiento a partir de datos ya existentes. Permite la introducción de datos no observados o la introducción de variables inferidas mediante modelos matemáticos. Es muy provechoso cuando los datos que se entrenan son complejos y el volumen de datos muestreados es pequeño. El aumento de datos soluciona el problema de los datos limitados y el sobreajuste del modelo.

6.7.2.4 Agregación de datos

La agregación de datos es una técnica que permite reducir el volumen de datos mediante la agrupación. Esta agrupación suele ser de un solo atributo. Por ejemplo, si se tiene un conjunto de datos con el atributo tiempo organizado en días a lo largo de una serie temporal determinada, se pueden agregar los datos en grupos mensuales, lo que facilita el manejo del atributo tiempo. Esto ayuda a reducir la amplitud de un atributo determinado sin pérdidas tangibles durante la manipulación futura de los datos.

6.7.3 Limpieza de los datos

También conocida como depuración de datos, permite detectar y corregir errores e inexactitudes en los datos recopilados. Se espera que los datos sean consistentes con los requisitos de entrada del modelo de ML. Las principales actividades de la limpieza de datos incluyen el ajuste fino de los datos con ruido y la gestión de los datos faltantes. Esto ayuda a garantizar que el dataset recopilado sea completo y que se elimine cualquier error o sesgo. También incluye la detección de valores atípicos dentro del dataset, tanto numéricos como no numéricos.

6.7.3.1 Análisis Exploratorio de Datos

Se basa en la información que se puede obtener de los datos recopilados y, en ocasiones, implica la visualización de datos. Esta visualización permite comprender las propiedades de los datos, como la asimetría y los valores atípicos. Este análisis se realiza principalmente con estadísticas. Las gráficas permiten comprender la distribución del dataset y el resumen estadístico de todos los atributos. El análisis exploratorio de datos (EDA) permite tomar decisiones, como las técnicas de limpieza de datos que se utilizarán, las transformaciones de datos necesarias, si es necesario reducir los datos.

6.7.3.2 Datos faltantes

Si bien es importante verificar durante la recopilación de datos que todos los atributos del dataset tengan un valor real, a veces hay valores faltantes, lo que dificulta su uso. Por ello, se han descrito diferentes técnicas para gestionar los datos faltantes. Las plataformas de manipulación de datos, como Python y R, incorporan algunas librerías que incluyen estas técnicas. La técnica suele variar según el conjunto de datos; por lo tanto, tras la evaluación de los datos en el análisis exploratorio, se puede seleccionar fácilmente la técnica más adecuada para la asignación de datos faltantes.

6.7.3.2.1 Asignación Deductiva

Sigue las reglas básicas de la lógica y, por lo tanto, es la asignación más sencilla, aunque también la que requiere más tiempo. Aun así, sus resultados suelen ser muy precisos. Por ejemplo, si los datos de los estudiantes indican que el número total de estudiantes es 10 y el número total de exámenes es 10, pero hay un examen sin nombre y Juan no tiene calificaciones registradas, la lógica dicta que el examen sin nombre es de Juan.

6.7.3.2.2 Asignación por Media/Mediana/Moda

Utiliza técnicas estadísticas donde se calculan las medidas centrales de tendencia en un atributo determinado y los valores faltantes se reemplazan con la medida de tendencia central calculada, ya sea la media, la moda o la mediana de ese atributo. Se aplica en datasets numéricos y afecta la fuente de datos inicial y el resultado del modelo. Las plataformas de manipulación de datos como Python y R tienen técnicas integradas para tratar datos faltantes.

6.7.3.3 Datos con Ruido.

La presencia de datos con ruido puede tener un efecto sustancial en el resultado de un modelo de ML, afecta negativamente la predicción del modelo. Los datos con ruido incluyen información innecesaria, valores redundantes y valores duplicados o inútiles. Esto se debe a errores en la recopilación de datos, problemas derivados de la entrada de datos, problemas derivados de las técnicas de transferencia de datos, convenciones de nomenclatura desiguales y a restricciones tecnológicas, en el caso de los datos no estructurados. . Los datos con ruido se eliminan mediante: el suavizado del dataset, la regresión lineal y el agrupamiento de datos.

6.7.4 Transformación de los datos.

La transformación de datos implica cambiar los datos depurados de un formato a otro, de una estructura a otra, o modificar los valores del conjunto de datos depurados para cumplir con los requisitos del modelo de aprendizaje automático. La simplicidad de la transformación de datos depende en gran medida de los datos de entrada necesarios y del conjunto de datos disponible. Incluye, la normalización, la selección de atributos, la discretización.

6.7.5 Reducción de los datos

Con el avance de las tendencias en tecnologías de la información y el crecimiento exponencial del IoT, se ha producido un aumento vertiginoso en el volumen de datos disponibles. Si bien, supone una gran ventaja para el ML, ya que la disponibilidad de big data para el entrenamiento de los modelos garantiza la precisión de la información generada por estos, la gestión y el análisis de estos enormes volúmenes de datos supone un gran reto, de ahí la necesidad de técnicas de reducción de datos. La reducción de datos reduce los costos de analizar y almacenar volúmenes de datos al aumentar la eficiencia del almacenamiento. Entre las técnicas se incluyen: el uso de cubos de datos y la selección de las características.

6.7.6 Sesgos en la preparación de los datos

El sesgo en los datos que entrenan en el modelo de ML hace predicciones erróneas. Es fundamental identificar en la fuente cualquier sesgo en el dataset durante la preparación y eliminarlo. El sesgo de muestra se produce durante la recopilación de datos, cuando la muestra seleccionada no representa correctamente a la población en estudio. Por ejemplo, un sistema de reconocimiento de iris entrenado exclusivamente con iris de personas africanas no identificará eficazmente los ojos de la población blanca.

El sesgo de exclusión es común en la etapa de limpieza de datos, donde se elimina o se tergiversa una parte de los datos, lo que lleva a su exclusión del entrenamiento del modelo. El sesgo de medición se produce durante el etiquetado de datos, donde un etiquetado no uniforme da lugar a predicciones erróneas del modelo de aprendizaje automático. El sesgo de recuperación también se produce en la etapa de etiquetado de datos, cuando el etiquetado no es consistente.

Otro es el sesgo racial, y el mejor ejemplo de este sesgo en los motores de búsqueda de voz, donde el modelo se entrenó principalmente con datos de voz de la población blanca y, por lo tanto, apenas reconoce la voz de la población negra [19]. El sesgo de asociación ocurre cuando un conjunto de datos ha creado una asociación implícita entre atributos. El principal sesgo de asociación es el sesgo de género, como en el caso en que un sistema se entrena con todos los directores de escuela hombres, lo que finalmente descalifica la plausibilidad de una directora de escuela.

6.8 Data sets en ML

Un data set es una colección de datos utilizada para crear, probar, afinar y en general trabajar modelos de ML. Cada elemento o registro de un data set, actúa como un ejemplo para enseñarle al algoritmo de ML cómo debe predecir un resultado cuando en producción se le entra un punto de datos. Los propósitos básicos de un data set son para entrenar un modelo y para medir su precisión cuando ya está entrenado. A este data set se le conoce como benchmark.

6.8.1 Partición del Data set en ML

En el momento de entrenamiento de los modelos, el dataset generalmente se particiona en tres conjuntos: el conjunto de entrenamiento, el conjunto de prueba y el conjunto de validación.

Conjunto de entrenamiento. Es el más importante y se compone de alrededor del 60% del dataset. Comprende los datos que serán utilizados para entrenar el modelo, en otras palabras le indica al algoritmo lo que debe buscar en los datos.

Conjunto de validación. Se usa para comprender el comportamiento de un modelo determinado a través de diferentes tipos de modelos. Permite darle validez al modelo en estudio. Cuando la precisión del conjunto de validación es mayor que la precisión obtenida con el conjunto de entrenamiento, estamos hablando de que el modelo esta sobre-entrenado.

Conjunto de prueba. Se usa para aproximar la precisión imparcial del modelo y darle una evaluación objetiva. Se busca un conjunto de datos del mundo real que todavía el modelo no conoce.

6.8.2 Data set abiertos

Son data sets listos para utilizar, que están disponibles, gratis, para descargar, utilizar, modificar y distribuir, sin restricciones legales o financieras. Para tipos de datos más específicos, ver https://research.aimultiple.com/datasets-for-ml/

Data sets abiertos públicos

Son conjuntos de datos de proyectos gubernamentales, disponibles al público. Pueden incluir censos, datos demográficos y otros. En Colombia, la Ley 1712 de 2014 de la Ley de Transparencia y del Derecho de Acceso a la Información Pública Nacional, establece la obligatoriedad de las entidades públicas de aperturar datos y define los datos abiertos en el numeral sexto como "*todos aquellos datos primarios o sin procesar, que se encuentran en formatos estándar e interoperables que facilitan su acceso y reutilización, los cuales están bajo la custodia de las entidades públicas o privadas que cumplen con funciones públicas y que son puestos a disposición de cualquier ciudadano, de forma libre y sin restricciones, con el fin de que terceros puedan reutilizarlos y crear servicios derivados de los mismos*".

6.9 Lenguajes y librerías

Para el desarrollo de aplicaciones ML, es preciso tener conocimientos de programación, de estructuras de datos, de manejo de memoria y mínimo de un lenguaje de programación.

Además, hay muchas librerías que se ofrecen para diferentes lenguajes de programación para hacer ML, por lo que es muy fácil para personas con los conocimientos básicos señalados.

Por otra parte, en concepto de otros expertos, no hay un mejor lenguaje para ML, cada uno es bueno en determinado campo. Por ejemplo, la mayoría de los ingenieros de ML prefieren usar Python para problemas de lenguaje natural NLP, mientras que para análisis de sentimientos prefieren usar Python o R, y algunos probablemente van a usar Java, para otras aplicaciones de ML como seguridad y detección de amenazas.

Python

En estos momentos hay más de 8 millones de desarrolladores alrededor del mundo que usan Python. Según Spectrum de la IEEE, es el primer lenguaje utilizado, en este último año. Pero hay más: es fácil de aprender, hay una comunidad madura que brinda soporte y en mi concepto lo más importante, hay cientos de librerías que permiten desarrollos rápidos en muchos temas de IA.

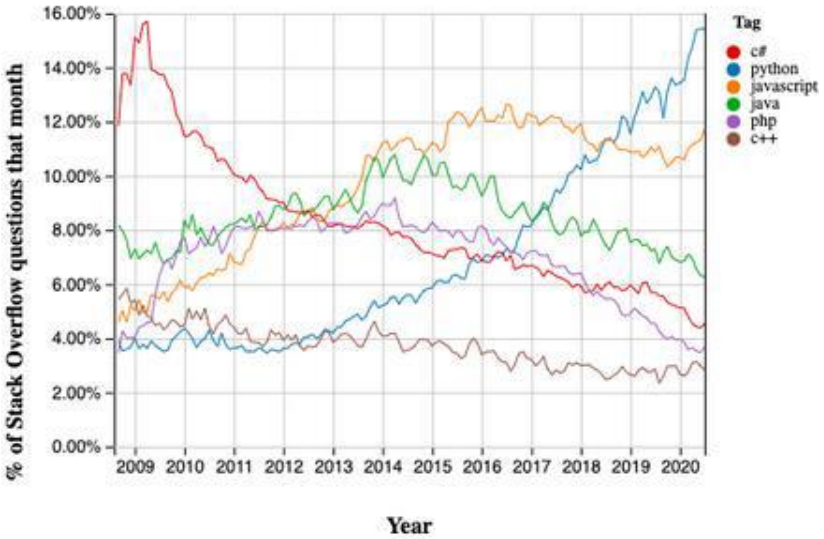


Figura 6.13. Uso de lenguajes de programación. Fuente Stack Overflow

R

Con más de dos millones de usuarios, 12000 paquetes en el repositorio CRAN, aproximadamente 206 grupos Meetup, más de 4000 preguntas sobre programación R cada mes, R es un lenguaje para escribir ML de estadísticas para estadísticos.

6.10. Proyectos de ML

Para el desarrollo de un proyecto de ML, se deben considerar los siguientes pasos:

1. Definir el objetivo del proyecto
2. Definir las características, o features, o variables del proyecto
2. Definir el criterio de evaluación
3. Evaluar la solución actual

4. Obtener un data set confiable y con los datos suficientes
5. Seleccionar el modelo
6. Analizar los errores del modelo
7. Aplicar el modelo para el objetivo planteado

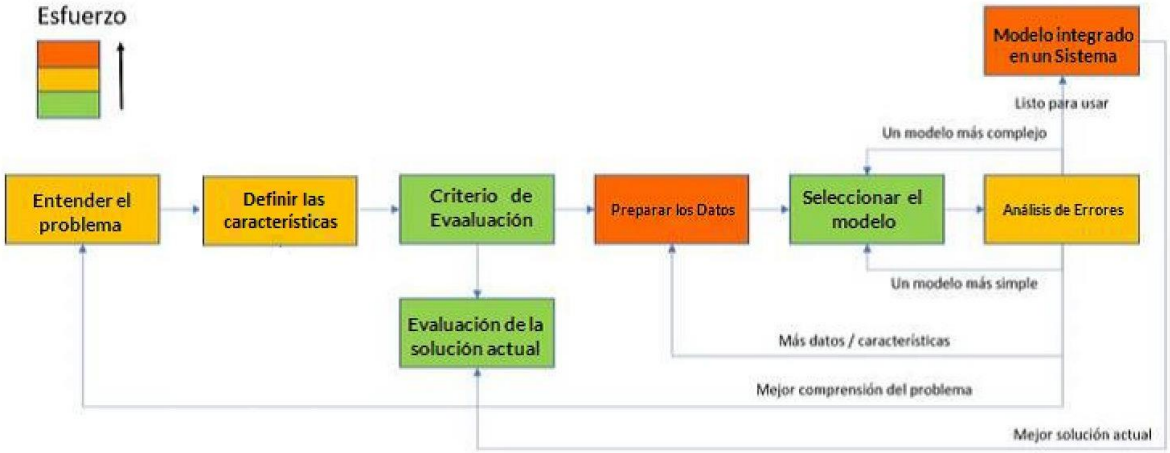


Figura 6.14, modelo para la implementación de sistemas de ML

6.11. ML oscura

Ahora más que nunca, los avances tecnológicos en ML nos traen soluciones revolucionarias a muchos problemas. Con todo lo positivo de estos desarrollos, también es necesario tener en cuenta los matices de estos avances que a veces hacen daño, en lugar de ayudar.

Un modelo de ML con sesgo, comienza a presentar resultados con prejuicios sistémicos como sesgos demográficos, culturales y estéticos. Bajo el paraguas de la IA y el ML, se utilizan dasets de entrenamiento para "enseñarle" a las máquinas sus funciones predictivas. Finalmente, la raíz de este problema es un error humano; los data sets incompletos llevan a predicciones imprecisas. Sin un amplio rango de representación de los diversos grupos en el mundo que utilizan internet, los productos digitales están sujetos a reflejar sesgos cognitivos no deseados, que señalan culturas, identidades, …, específicas únicas.

Veamos un caso en Amazon, con experiencia en IA, donde ha habido problemas. Según un reporte de Reuters, este gigante del e-comerce tenía un sistema interno que usaba IA para examinar las aplicaciones, que hacen las personas interesadas en trabajar con Amazon. Después de algunas quejas se encontró que una herramienta de reclutamiento de personal tenía perjuicios contra las mujeres degradándolas consistentemente. Este sistema fue descartado por Amazon, luego de observarse estos comportamientos.

Como se ha visto, los sistemas de ML funcionan con datos históricos, en los que se han mantenido estos prejuicios, generalmente llevando a que se perpetúen los prejuicios existentes. En este caso el prejuicio estaba en los ambientes de trabajo dominados por hombres.

Otra manera en que se implementan prejuicios de género y raza se debe a cómo, de manera inconsciente, los desarrolladores hacen las implementaciones con estos sesgos. Este es un problema

muy complicado de resolver y la única solución es a través de una muy buena educación, consideración que las empresas de desarrollo en general no son conscientes.

En su libro, Cathy O´Neal, “Armas Matemáticas de Destrucción”, plantea que estamos viviendo en la edad del algoritmo, que las decisiones que afectan nuestras vidas no las toman los humanos sino los modelos matemáticos. En teoría esto lleva a una mayor equidad, pues en principio, todos somos juzgados por las mismas reglas. Pero en la práctica ocurre todo lo contrario, los modelos son opacos, no están regulados, son incontestables. Esto lleva a un refuerzo en la discriminación.

Supongamos que ustedes se gradúan, generalmente van a trabajar en el desarrollo de un proyecto. En poco tiempo ven la necesidad de salir de la casa y adquirir un apartamento pequeño. Van al banco donde le consignan su salario y manifiestan su interés de compra con el fin de conocer todas las condiciones para que le presten para una hipoteca. Y comienzan los problemas, si bien su ingreso es alto, no tiene contrato a término indefinido, no tiene experiencia crediticia y otros. Las reglas de la banca actual no permiten que se le otorgue un préstamo, estas reglas se llevan a los algoritmos, perpetuando la discriminación.

En el libro “THE BOOK OF WHY”, j Pearl y D. Mackenzie, plantean el problema de ML en el sentido que llevan a que características (features) altamente correlacionados se asumen como relaciones causa efecto, llevando a veces a resultados completamente desatinados. Sin embargo, es importante destacar el problema de que la correlación no implica causa.

Por ejemplo, si se hiciera una gráfica de atracos en Colombia contra el número de ingenieros mecatrónicos, se tendría una correlación espuria y se construyera un modelo entrenado con un dataset de este tipo, simplemente no funcionaría ya que el modelo no entiende las causas detrás de las relaciones entre los datos, por lo que se hace necesario que entendamos la pregunta del porqué de estas relaciones. Y es aquí donde nace una ciencia poco conocida, aunque también cada vez más en auge, como es la **inferencia causal.**

La inferencia causal consiste en el proceso de identificar y entender las relaciones causa-efecto entre variables. En muchas áreas de investigación, incluyendo la epidemiología, la economía, la ciencia política y la psicología, la inferencia causal es crucial para entender los efectos de las intervenciones, políticas o programas. La inferencia causal también ayuda a predecir el resultado de cambios en las variables, lo que puede ser especialmente útil en el diseño de experimentos y en la toma de decisiones. De manera que en modelos de ML se debe considerar hacer este tipo de análisis para darles una mayor validez

6.12. Ejercicios

1. Considere la figura 6.1. Conforme un data set con resultados de 50.000 multiplicaciones de matrices 2 X 2, generadas aleatoriamente con números enteros entre -20 y 20. Una vez entrenado el modelo, úselo con 10 ejemplos y compare los resultados con los que se producen al realizar la operación analíticamente. Haga un cálculo de que es más costoso computacionalmente.

En alguno de los siguientes problemas, usar datos de alguna entidad gubernamental.

2. Estudie el algoritmo SVM con todo detalle, mejore su documentación y con base en el haga cambios para una aplicación.

3. Estudie el algoritmo de K- Nearest, con todo detalle mejore su documentación y con base en el haga cambios para una aplicación.
4. Estudie el algoritmo de árboles de decisión, con todo detalle mejore su documentación y con base en el haga cambios para una aplicación.
5. Estudie el algoritmo de Bayes ingenuo y haga un ejemplo bien documentado.
De los últimos 4 ejercicios haga 3.
Nuevamente, si tiene otro problema que le parezca interesante en el que se use ML, lo puede cambiar, por uno de los ejercicios planteados.

6.13 Bibliografía

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, Concepts, Tools, and Techniques to Build Intelligent Systems, **Aurélien Géron.**
Machine Learning in Robotics, 5 Modern Applications, Daniel Fegguella, https://emerj.com/ai-sector-overviews/machine-learning-in-robotics/
Cathy O´Neal, Armas Matemáticas de Destrucción, 1972
J. Pearl y D. Mackenzie, THE BOOK OF WHY, The New Science of Cause and Effect, Basic Books, 2018.
Laurence Moroney, AI and Machine Learning for Coders, O'Reilly Media, 2020
Data preparation for ML, Rachael Njeri Ndung'u, Murang'a University College, 2020

7. INTELIGENCIA ARTIFICIAL GENERATIVA

José J. Martínez P.

Marzo 2026

7.1. Introducción

Está por demás hablar del impacto que produjo la liberación de ChatGPT el 30 de noviembre de 2022. Hablamos entonces de que los avances en IA en el área ahora conocida como IA Generativa, están marcando nuevos derroteros en la disciplina de la IA. Se dice que hay un antes y un después de 2018 cuando se desarrolló el concepto de "Transformer", como una nueva estructura de ML para la generación de grandes modelos de lenguaje, MLLs.

Pero, lo más impactante, no solo fue el desarrollo de esta nueva tecnología, sino su utilización casi inmediata por decenas de millones de personas sin formación científica o tecnológica, distribuidas en muchas partes del mundo. Es la primera vez en la historia de la humanidad que se presenta este suceso, la utilización inmediata a escala planetaria de una nueva tecnología.

Desde los 80s se comenzó a trabajar en ML, especialmente enfocado hacia el reconocimiento de objetos o patrones, ***con base en el concepto de características*** (features), sin embargo, con Deep Learning de alguna manera se dispara su utilización, llevando al reconocimiento de los datos, como la fuente de conocimiento, pues en ellos está la información real, válida, de la interacción de humanos y máquinas con el ambiente, todo esto considerando las RNAs como su columna vertebral.

Los desarrollos que han conllevado a la reciente aparición de herramientas de IA Generativa, también han llevado a replantear algunas consideraciones sobre el ML. ¿Qué es IA? ¿Cuál es la diferencia entre IA y ML? ¿Cuál es la diferencia entre ML y DL?

Esencialmente la IA, es una disciplina que tiene que ver con la teoría y los métodos para construir máquinas que piensen y actúen como los humanos. Revisando las definiciones tenemos que:

IA: Busca aprovechar los computadores y las máquinas para imitar las capacidades de solución de problemas y toma de decisiones de la mente humana.
ML: Es un subconjunto de la IA, que se enfoca más en el uso de varios algoritmos de auto-aprendizaje que obtienen conocimiento de los datos, con el fin de predecir resultados.
DL: Es un subconjunto de ML. Generalmente se piensa como una máquina de aprendizaje debido a que automatiza y profundiza mucho el proceso de extracción de características y elimina algunas intervenciones humanas, que nos permite el uso de algunos, realmente grandes, conjuntos de datos que pueden o no estar estructurados. En la figura 1, podemos ver esa relación.

.

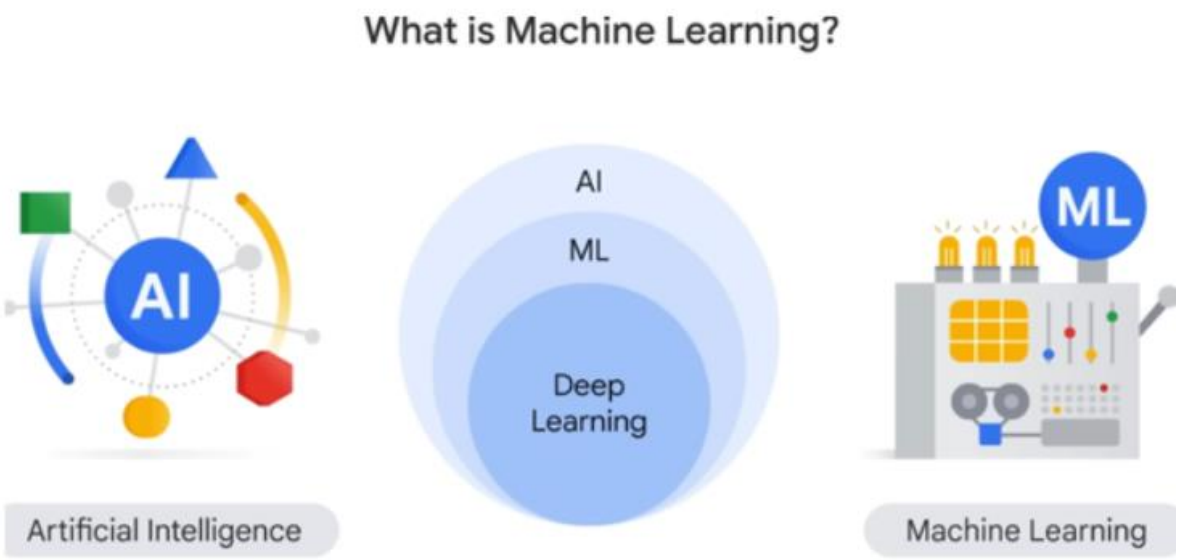


Figura 7.1. Relación entre IA. Ml y DL.

7.2. IA Generativa

La IA Generativa es un tipo de tecnología de IA que puede producir diferentes tipos de contenido incluyendo texto, imágenes, audio, video y datos sintéticos; y lo hace a partir de patrones aprendidos de datos existentes. La IA generativa es capaz de producir contenido altamente realista y complejo que imita la creatividad humana, a partir de indicaciones (prompts) hechas por personas que pueden no tener ningún conocimiento de tecnología, convirtiéndose en una herramienta valiosa en muchos campos del quehacer humano.

La historia de la IA Generativa, puede decirse que comenzó en la década de los 50s y 60s cuando los primeros investigadores comenzaron a explorar las posibilidades de la IA, enfocados en desarrollar sistemas basados en reglas que podían simular el pensamiento humano y la toma de decisiones. En los 90s emergieron las RNAs, inspiradas en el cerebro humano, capaces de aprender de los datos en una forma que no lo hacen los sistemas basados en reglas. Esto llevó a grandes avances.

Uno de los avances más significativo fue el desarrollo del Deep Learning, que permitió a la IA Generativa lograr nuevos niveles de realismo y creatividad en el último par de años, que se pueden atribuir a las siguientes tres razones:

Disponibilidad de datos. En el pasado los modelos de IA Generativa estaban limitados por la cantidad de datos de los cuales aprender. Ahora, con la disponibilidad de miles de millones de datos estos modelos aprenden de miles de millones de ejemplos, volviéndolos más precisos y realistas.

Avances en la capacidad de cómputo. La IA Generativa requiere mucho poder de cómputo para su entrenamiento, por lo que su disponibilidad ha permitido entrenar los modelos con data sets mucho más grandes con mejoras en la precisión y el realismo.

Nuevos algoritmos. Los investigadores han desarrollado nuevos algoritmos específicamente para IA Generativa, mucho más efectivos que otros algoritmos y procesos, lo que ha llevado a mejoras recientes en IA Generativa.

6.3 Comportamiento de las herramientas de IA Generativa

Para evaluar el comportamiento de las herramientas de IA Generativa se utilizan benchmarks. Un benchmarck o prueba de referencia, es una medida de calidad que permite evaluar un desempeño contra otro. Así para evaluar el comportamiento de herramientas de ML existen una serie de data sets, benchmarks, de los cuales uno de los primeros fue MNIST ya mencionada, con números escritos a mano.

En un artículo de 2021, de Pott, Kiela et al., se presenta un gráfico de cómo la IA Generativa ha ido superando diferentes benchmarks, comparándola con el comportamiento humano para superar esos benchmarks. El gráfico parte de los 90s en el eje X; y en el eje Y, toma una medida normalizada del estimado del comportamiento humano que es la línea roja, en cero.

Así el reconocimiento de los números escritos a mano de MNIST que se lanzó en los 90s, tomó aproximadamente 20 años en sobrepasar el estimado del comportamiento humano. Switchboard tiene una historia similar, se lanzó en los 90s, su problema es pasar de voz a texto; tomó aproximadamente 20 años pasar la línea roja. ImageNet tiene una historia más reciente, se lanzó en 2009, tomó 10 años alcanzar el punto de saturación; y a partir de aquí realmente se ha acelerado el paso de la línea roja, así SQUAD 1.1, para responder preguntas, se solucionó en 3 años. La respuesta fue SQUAD 2.0, que se solucionó en menos de 2 años.

Y luego el benchmark GLUE, que es un conjunto grande de tareas destinado a probar el estrés de los mejores modelos. Cuando se anunció a muchos de los científicos del campo les preocupó que fuera muy difícil para los modelos actuales. Pero GLUE se saturó en menos de 1 año. La respuesta fue SuperGlue destinado a ser mucho más difícil y también se saturo en menos de 1 año.

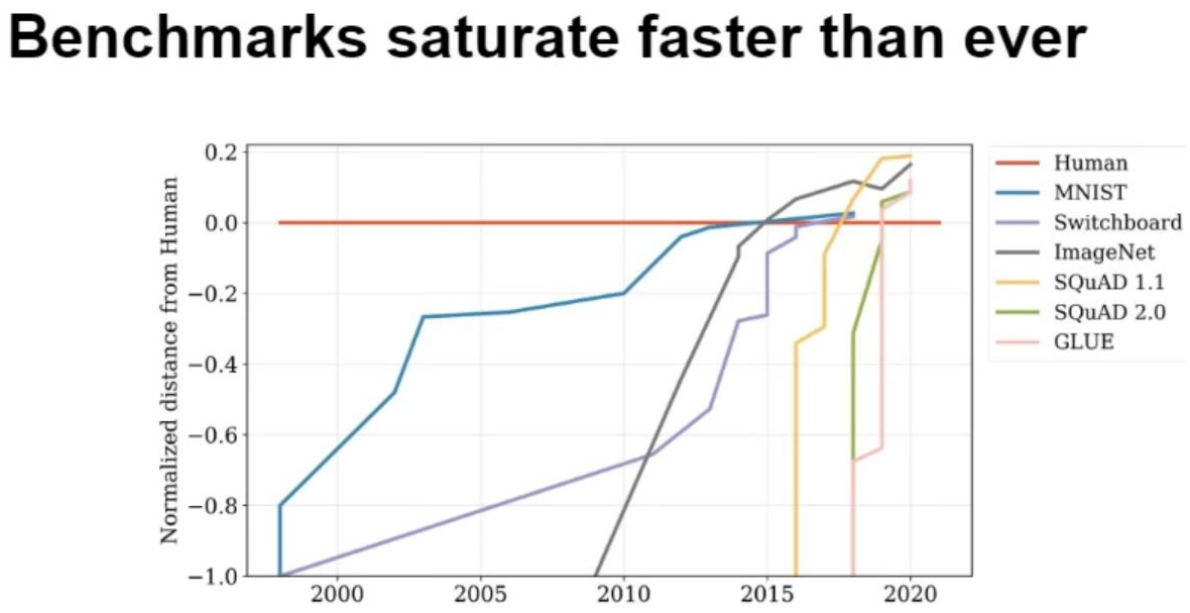


Figura 7.2. Cada vez es más rápido que las nuevas herramientas de IA Generativa, saturen los benchmarks.

Esta es una historia notable de progreso indudable, aun si se tienen dudas sobre las medidas del comportamiento humano, incluso aquí vemos un rápido incremento en la tasas de cambio. Y ahora 2021 fue hace años en la historia de la IA, específicamente en términos de NPU (Natural Language Understanding).

7.4. Modelos Fundamentales.

Los grandes modelos de lenguaje MLL como ChatGPT, son realmente una parte de una clase diferente de modelos llamada modelos fundamentales. El término fue acuñado por el grupo de investigación de NPL en Stanford, cuando se dieron cuenta que este campo de la IA se estaba convirtiendo en un nuevo paradigma.

Las aplicaciones de IA estrechas se construyen entrenado cada una con data sets muy específicos del dominio, produciendo diferentes modelos de ML, cada uno para una tarea específica.

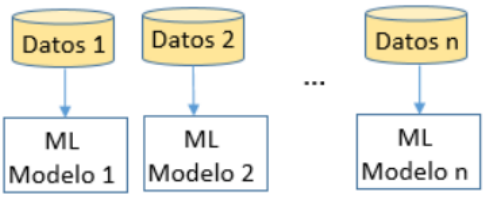


Figura 7.3. Se tienen aplicaciones de IA estrecha, para propósitos específicos, cada una con su propio data set, produce con algún algoritmo de ML, un modelo especifico.

Se predijo que estábamos yendo a un nuevo paradigma, donde tendríamos una capacidad fundamental o modelo fundamental, que podría llevar todos estos mismos casos de uso y aplicaciones. Así, las mismas aplicaciones que se proveían con estos modelos, podrían ahora construir los mismos modelos y un número de aplicaciones adicionales. El punto es que a este nuevo modelo se le puede transferir cualquier número de tareas. Lo que lleva a que este modelo tiene el superpoder de ser capaz de transferirse a múltiples tareas y efectuar múltiples funciones.

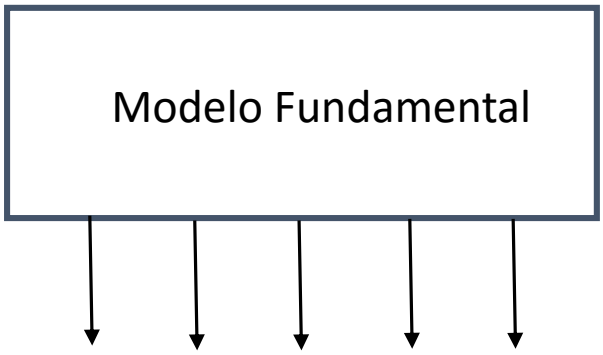


Figura 7.4. Modelo fundamental, incluye todos los data sets de todos los modelos estrechos, más datos no rotulados. Puede entonces, dependiendo de la afinación producir los modelos de IA estrecha, además de muchos otros.

Un modelo fundamental es un algoritmo de DL pre-entrenado con data sets extremadamente grandes obtenidos de internet. No son como los modelos de IA estrechos, entrenados para efectuar una sola tarea, pues se entrenan con una amplia variedad de datos y pueden transferir conocimiento

de una tarea a la otra. Este tipo de RNA se entrena una vez y luego se afina para realizar diferentes tipos de tareas.

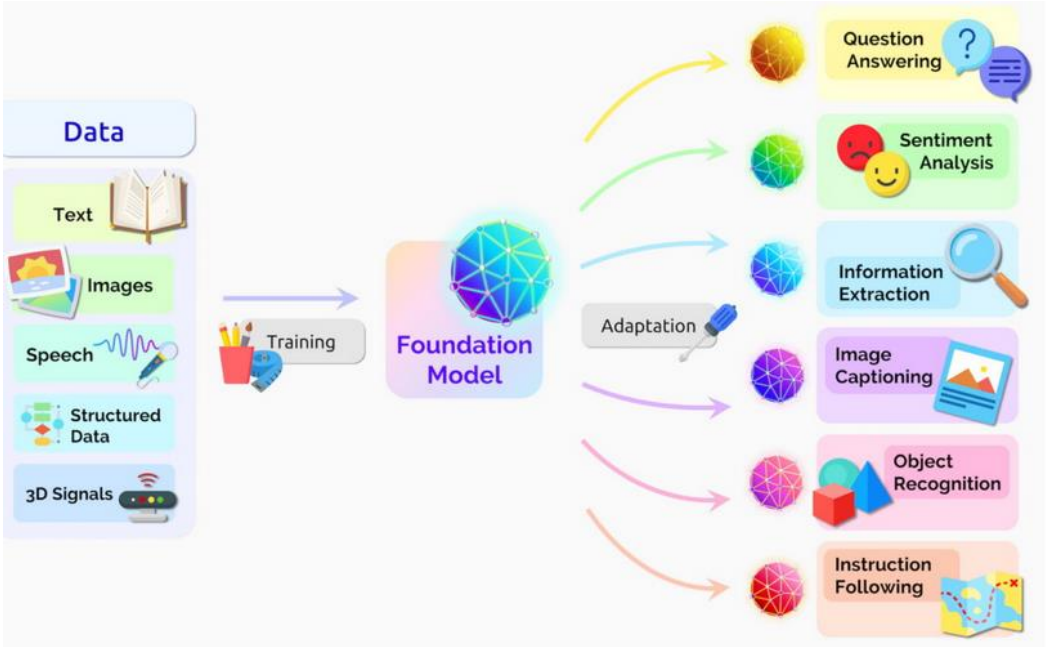


Figura 7.5. Construcción y utilidad de un modelo fundamental.

Estos modelos, figura 7.5, cuestan millones de dólares debido a que contienen cientos de miles de millones de hiperparámetros que se han entrenado con cientos de gigabytes de datos. Sin embargo, una vez terminado, cada modelo fundamental se puede modificar un número ilimitado de veces para automatizar un número ilimitado de tareas discretas.

Los modelos fundamentales actuales se usan para entrenar aplicaciones de IA que se basan en procesamiento de lenguaje natural. NLP, y la generación de lenguaje natural, NLG. Su uso más popular incluyen Claude, GPT-4, DeepSeek y DALLE-E 2.

7.5. Transformers

Los modelos fundamentales utilizan la arquitectura Transformer que es un tipo de arquitectura de RNA que ha ganado mucha popularidad en el campo de la IA Generativa. El desarrollo de los transformers estuvo dirigido a abordar el problema de la generación de una secuencia de texto, que incluye tareas como traducción, reconocimiento de voz, conversión texto a voz y más. Pero antes de entrar en detalle, veamos un par de conceptos claves para una mejor comprensión de la arquitectura Transformer.

Token

Un token es una unidad de datos que viene de descomponer fragmentos más grandes de información, que puede representar palabras, caracteres o frases. Cuando se procesa texto, una frase se divide en tokens, donde cada palabra o signo de puntuación se considera un token separado. El proceso de tokenización es crucial en la preparación de los datos para el proceso

posterior en modelos de IA. Los tokens no están restringidos solo a texto, pueden representar diferentes formatos de texto y juegan un papel fundamental en la habilidad de los LLMs para aprender y comprender.

Los tokens son fundamentales para que los LLMs interpreten la entrada y predigan la salida, manteniendo el contexto dentro de una ventana fija. El proceso típico incluye:

- **Tokenización.** El modelo divide el texto de entrada en tokens. Este proceso es una parte básica del procesamiento de lenguaje natural.
- **Tipos de tokens**
  - Tokens de texto. Se usan en los LLMs para generar respuestas similares a las de un humano, en diferentes aplicaciones.
  - Tokens de imágenes. Se aplican en modelos como Dall-e y Stable Diffusion donde se dividen las imágenes en estructuras similares, para generación de arte.
  - Tokens de audio. Se utilizan en modelos de voz de IA, donde las palabras habladas se convierten en representaciones tokenizadas para procesamiento y generación.

La mayoría de los tokens se manejan para mejorar las capacidades de estos modelos, permitiendo un procesamiento y generación más eficiente del lenguaje humano.

Los modelos de IA procesan tokens para aprender las relaciones que hay entre ellos y desbloquear capacidades de predicción, generación y razonamiento. Entre más rápido se puedan procesar los tokens, los modelos pueden aprender y responder más rápido.

Importancia de los tokens

Costo y eficiencia. El costo de uso de los MLLMs, en su mayoría, se basan en el uso de tokens, entre más tokens, más costo computacional. Comprender los límites de los costos es crucial, pues afectan directamente los gastos operativos.

Longitud del contexto

El número de tokens que puede procesar un LLM afecta su habilidad para mantener el contexto y la coherencia en la respuesta.

Ajuste fino y entrenamiento

Los desarrolladores optimizan los modelos ajustando la forma en que se van procesar los tokens, para mejorar su fluidez y relevancia. Son los ladrillos de construcción de los LLMs.

Ejemplo de tokenización.

```
Texto original:
Hola, estoy probando la tokenización como lo hace ChatGPT.

Tokens (IDs numéricos):
[69112, 11, 82384, 3650, 4988, 1208, 4037, 42600, 8112, 781, 35905, 13
149, 38, 2898, 13]
```

```
Número de tokens: 15

69112  --> 'Hola'
11     --> ','
82384  --> ' estoy'
3650   --> ' prob'
4988   --> 'ando'
1208   --> ' la'
4037   --> ' token'
42600  --> 'ización'
8112   --> ' como'
781    --> ' lo'
35905  --> ' hace'
13149 --> ' Chat'
38     --> 'G'
2898   --> 'PT'
13     --> '.'
```

Figura 7.6. Tokenización estilo GPt

```
Texto reconstruido:
Hola, estoy probando la tokenización como lo hace ChatGPT.
```

Funcionamiento de los tokens:

1. Diccionario fijo.

Cada modelo, por ejemplo GPT-2, GPT-3, GPT-4 tiene su propio diccionario de tokens, que asigna a cada palabra, subpalabra o símbolo, un número entero único.

2. Universalidad dentro del modelo

Esto asegura que el mismo texto siempre se convierte en la misma secuencia de tokens, algo esencial para que el modelo pueda aprender y predecir.

Embeddings

Son representaciones vectoriales numéricas de datos complejos como texto, imágenes o audio, que usan los LLMs para comprender significados y relaciones. Al transformar estos datos brutos en números de punto flotante de más baja dimensionalidad, los embeddings capturan significado semántico y relaciones estructurales, permitiendo que los LLMs procesen, comparen y encuentren elementos similares, eficientemente.

El embedding de un token se obtiene al pasar su número por la capa de embedding del modelo, que consulta su matriz de embeddings, creando un vector con una determinada dimensión: 768, o 1024 u otra dependiendo del modelo, inicialmente único para cada token.

Cada componente de ese vector puede representar algo como:

- La palabra es un sustantivo o un verbo
- La palabra tiene connotación positiva o negativa
- La palabra se relaciona con un tema específico como política, deportes, economía, ciencia.

Al combinar todos estos componentes, el MLL, puede "comprender" mejor el significado y generar texto más coherente y relevante.

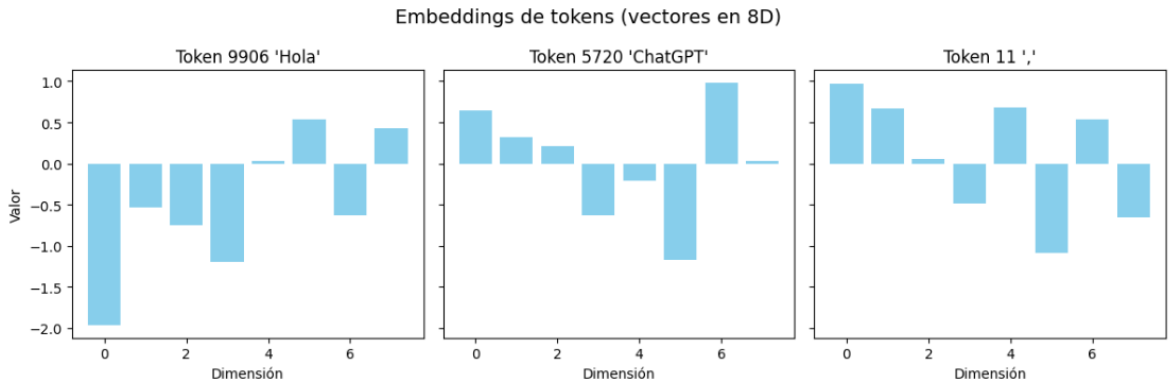


Figura 7.7. Vectores embedding de varias palabras

Sin embargo, los embeddings van evolucionando capa por capa para capturar el contexto y el significado de cada palabra.

En el artículo, "Attention Is All You Need" de T. Brown, et al, traducido, "El contexto es todo lo que usted necesita", se plantea la arquitectura Transformer, figura 7.8, que solo se basa en los mecanismos de atención pues no incluyen mecanismos de recurrencia y convolución. Los mecanismos de atención se han convertido en la parte integral del modelado de secuencias convincentes y transducción en varias tareas, pues permite el modelado de dependencias sin considerar sus distancias en las secuencias de entrada y salida.

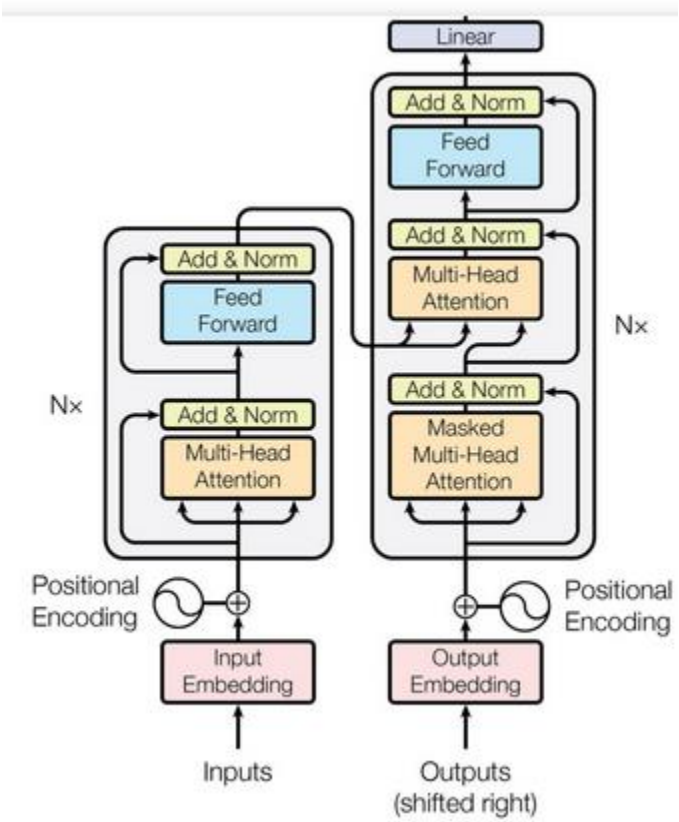


Figura 7.8. Arquitectura Transformer

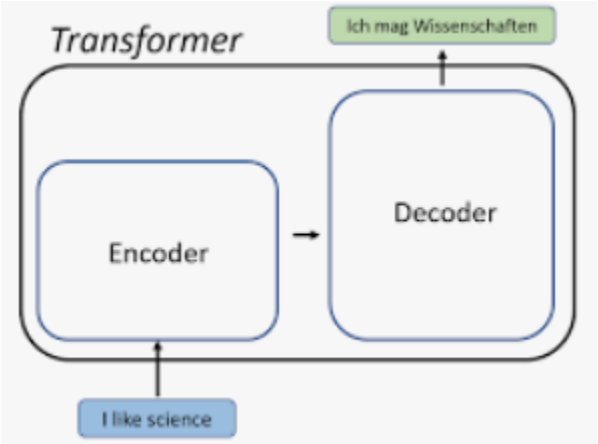


Figura 7.9. Arquitectura encoder-decoder

7.5.1 Encoder.

En un modelo transformer, el encoder, figura 7.9, permite al modelo comprender y codificar la secuencia de entrada, La arquitectura del encoder tiene varios componentes y procesos, con una pila de N= 6 capas idénticas:

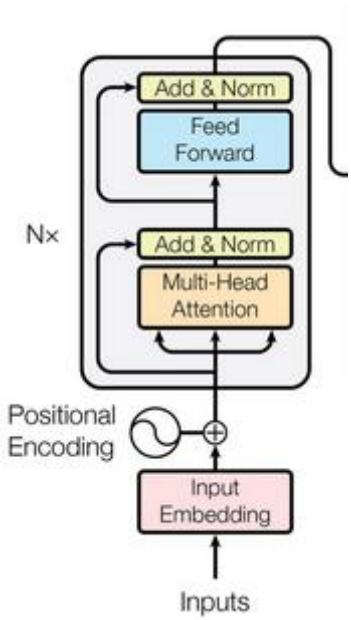


Figura 7.10. Arquitectura Encoder.

Embeddings. Los computadores solo comprenden datos numéricos, como vectores y matrices, por lo que es necesario convertir por ejemplo palabras, frases, imágenes, u otros, en vectores numéricos, generando un espacio vectorial, con contenido semántico. La idea, es que los elementos similares, en un dominio de aplicación específico, están muy cerca en ese espacio vectorial.

- Si hablamos de texto, el espacio embedding es un espacio donde cada palabra, a la cual se le ha asignado un valor, se agrupa con palabras similares. Por ejemplo, en el espacio vectorial, las palabras *rey* y *reina* estarán cerca, así como *parís* y *francia*.
- Si hablamos de imágenes, el embedding es un espacio de características visuales como forma, textura, color; lo que permite que el modelo comprenda conceptos visuales complejos.
- Si hablamos de audio y música, el embedding es un espacio de características como tono, ritmo, o timbre; lo que permite a los modelos generar música coherente.

Por ejemplo los embeddings de texto, permiten medir la relación entre las cadenas de texto. Comúnmente se usan para:

- Búsqueda. Los resultados se clasifican por la pertenencia a un texto de consulta
- Agrupamiento. Se agrupan cadenas de texto por similaridad.
- Recomendaciones. Se recomiendan ítems con cadenas de texto relacionadas.
- Detección de anomalías. Se identifican los valores atípicos porque tienen muy poca relación.
- Medidas de diversidad. Se analizan distribuciones de similaridad.
- Clasificación. Las cadenas de texto se clasifican por los rótulos más similares.

Encoder posicional. Aborda el problema de la ambigüedad en las palabras del prompt, proveyendo el contexto con base en la posición de las palabras dentro de una frase. Entonces cada token se convierte en un vector embedding aumentado con un embedding posicional. Estos embeddings crean un vector de contexto. El encoder posicional permite al modelo comprender el significado de las palabras en diferentes contextos de las frases, ampliando su habilidad para capturar relaciones y dependencias en la secuencia de entrada.

Multi-head attention. El aspecto central de la arquitectura transformer es su mecanismo de "auto-atencion", que determina la importancia de cada palabra con respecto a las otras palabras dentro de una frase. Esto se logra generando un vector de atención para cada palabra que captura las relaciones contextuales entre las palabras de la frase. El uso de múltiples vectores de atención se conoce como bloque de atención multi-head.

Vectores Q, K y V

La idea es que en lugar de usar vectores x_i, directamente se representa por tres papeles separados que juega cada vector x_i.

Q: Como el elemento corriente que se está comparando con las entradas precedentes.
K: Como una entrada precedente que se está comparando con el elemento corriente para determinar su similaridad
V: Un valor del elemento precedente que se pondera y se suma.

Si se tiene una secuencia de embeddings de tokens, X1, x2, x3, x4, x5, x6, x7 y x_i

Produce v_j = una suma ponderada de x1 a x7 incluyendo xi, ponderada por su similitud con xi

$$Score(x_i, x_j) = x_i . x_j$$
$$\propto_{ij} = softmax(Score)\ \forall_{j \leq i}$$

$$v_j = \sum_{j \leq i} \propto_{ij} x_j$$

Red feedforward

El cuarto paso incluye una red neuronal feedforward aplicada a cada vector de atención. La meta es transformar los vectores de atención en un formato que las capas subsecuentes del codificador o del decodificador puedan procesar. La red feedforward procesa los vectores de atención individualmente, uno a la vez. A diferencia de las Redes Neuronales Recurrentes, RNNs, los vectores son independientes uno de otro. Como resultado, se puede utilizar paralelización lo que mejora significativamente la velocidad de procesamiento y la eficiencia.

Con la habilidad para procesar los vectores de atención independientemente y en paralelo, podemos pasar todas las palabras al bloque encoder al mismo tiempo, produciendo un conjunto de vectores codificados para cada palabra, que se puede computar simultáneamente.

7.5.2 Decoder

La arquitectura del decoder, figura 4, lo mismo que la encoder, tiene varios componentes y procesos, con una pila de N= 6 capas idénticas. En un modelo transformer, el decoder juega un papel crucial en generar la secuencia de salida con base en la representación de la entrada codificada. Mientras que el encoder se enfoca en comprender la secuencia de entrada, el decoder se enfoca en generar la secuencia objetivo.

El ejemplo común es la traducción de una frase en ingles al francés. Se le provee una frase en inglés y la correspondiente traducción al francés para entrenar el modelo. La frase en inglés se procesa a través del bloque encoder, mientras que la frase en francés, la procesa el bloque decoder. Como el bloque encoder, el bloque decoder también incluye una capa embedding y un componente encoder posicional, que convierte las palabras de la frase de entrada en los vectores correspondientes. La arquitectura del decoder consiste de varios componentes y procesos que contribuyen a su funcionalidad como:

Masked multi-head attention

En este proceso, cada palabra de la frase interactúa dinámicamente con las palabras de su alrededor para descubrir las intrincadas relaciones entre ellas. El término "masked" se usa para evitar que el modelo se fije en las palabras que siguen durante este proceso. Continuando con el ejemplo, la secuencia de la entrada en francés se procesa a través del mecanismo de auto-atención. Este mecanismo es un componente fundamental que se usa tanto como vimos en el bloque encoder como en el bloque decoder, pues permite capturar las relaciones de dependencia entre diferentes palabras dentro de una frase o una secuencia.

Para profundizar en el mecanismo de aprendizaje, durante el proceso de entrenamiento, el modelo primero predice la traducción al francés de cada palabra utilizando sus resultados anteriores. Estas traducciones predichas luego se comparan con la traducción real en francés, que se proveyó como entrada al bloque decoder. Con base en la comparación, el modelo actualiza los valores de sus matrices, permitiéndole mejorar sus predicciones en cada iteración. Este proceso de aprendizaje iterativo continúa hasta que el modelo traduce ***precisamente*** las frases de entrada.

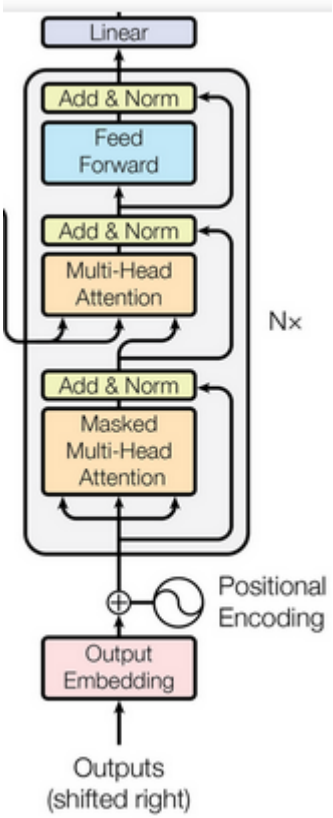


Figura 7.11. Arquitectura Decoder.

Durante el entrenamiento, se oculta o enmascara la siguiente palabra en francés. Esto asegura que el modelo debe aprender a traducir cada palabra al francés basado solamente en la palabra en inglés correspondiente y las palabras predichas al francés que lleva. Al enmascarar cada palabra en francés, evitamos que el modelo simplemente memorice los pares entrada-salida y lo anima a aprender patrones ocultos y relaciones entre los dos lenguajes. Cuando se procesa la frase en francés en el bloque decoder, solamente podemos usar las palabras previamente generadas en francés para predecir la palabra siguiente.

Par lograr esto, se enmascaran todas las futuras palabras en francés transformándolas en 0s. Esto se hace creando una matriz de máscara con la misma forma de la matriz de atención, con valores de 1 para la correspondiente palabra en francés que está disponible y 0 cuando esta enmascarada. Durante la operación de atención, se utiliza esta matriz de máscara para poner en 0 los pesos de atención, para las palabras enmascaradas en francés, asegurando que no contribuyen a la predicción de la palabra siguiente.

Multi-head attention block

El siguiente paso incluye pasar los vectores de atención obtenidos de la capa previa y los vectores codificados del bloque encoder a través de otro Multi-head attention block, donde se usan los resultados del bloque encoder. El bloque de atención encoder-decoder es donde sucede la principal proyección entre las palabras en inglés y en francés. Los bloques de atención que se generan en este bloque, capturan la relación contextual entre las palabras de la frase en inglés y aquellas en la correspondiente frase en francés.

Red feedforward

La unidad feedforward normalmente es una red neuronal simple de dos capas con una función de activación ReLu entre ellas. Se aplica a cada vector de atención independientemente, lo que permite la paralelización. Esta capa ayuda a transformar los vectores de atención en una forma más fácilmente digerible por la siguiente capa del modelo.

La salida de la capa lineal se pasa vía la capa softmax, produciendo una distribución de probabilidad sobre las posibles palabras de salida en francés. La función softmax, o función exponencial normalizada, es una generalización de la función logística, que se emplea para comprimir un vector K-dimensional Z, de valores reales arbitrarios, en un vector k-dimensional, $\sigma(Z)$, de valores reales en el rango [0, 1]. A cada palabra se le asigna una probabilidad con base en si la traducción reproduce fielmente la palabra inglesa introducida. Se escoge la palabra con más alta probabilidad como la palabra traducida para esa posición en la frase. Se repite este método para cada una de las palabras de la frase original en inglés para producir la frase traducida al francés.

7.6 Grandes modelos de lenguaje, LLM

Los LLMs son avances recientes en modelos DL para trabajar con lenguajes naturales humanos cuyo gran uso y utilidad se ha demostrado. Como humanos, percibimos el texto como una secuencia de palabras. Así, una frase es una secuencia de palabras, los documentos son una secuencia de capítulos, secciones y frases. Para los computadores, el texto es solo una secuencia de caracteres con valor numérico. Para que un computador pueda “comprender” un texto, se puede construir un

modelo con base en redes neuronales recurrentes. Este modelo procesa un carácter a la vez y provee una salida, una vez se haya procesado todo el texto. Este modelo trabaja bastante bien excepto algunas veces que “olvida” el comienzo de la secuencia cuando se llega al final del texto.

En 2017, Vaswani et al., publicaron el artículo, "Attention is All You Need", para establecer el modelo Transformer, que se basa en el mecanismo de atención. A diferencia de las redes neuronales recurrentes, el mecanismo de atención permite mantener toda la frase o incluso el párrafo a la vez, en lugar de una palabra cada vez. Esto permite al modelo Transformer comprender mejor el contexto de una palabra. Muchos modelos de PNL de última generación se basan en el modelo Transformer.

De acuerdo a C. Shannon, en su artículo “Prediction and Entropy of Printed English”, el lenguaje inglés tiene una entropía de 2.1 bits/letra, a pesar de que tiene 27 letras incluyendo el espacio. Si las letras se usaran aleatoriamente la entropía sería de 4.8 bits/letras (log2 27), lo que nos dice que en los textos y los audios, no hay tanta aleatoriedad. Esto permite hacer más fácil el manejo computarizado que de un texto de lenguaje humano. Los modelos de ML y especialmente los modelos Transformer son expertos en utilizar esta baja entropía para hacer las predicciones de texto.

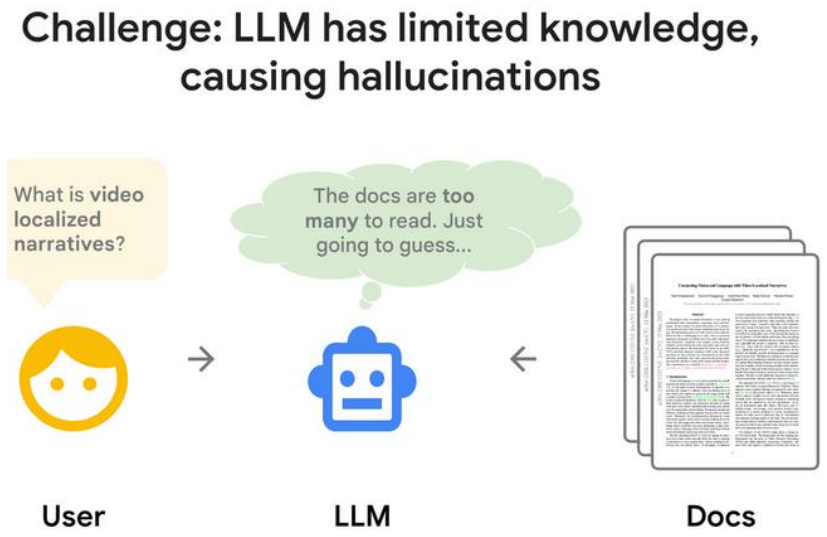
Pero, ¿cómo ve la gramática un modelo Transformer? Ya habíamos visto cómo es de complicado tratar de programar la serie de reglas de un lenguaje. En realidad el modelo transformer no almacena estas reglas, las adquiere implícitamente a través de ejemplos. Es posible que el modelo aprenda mucho más que reglas gramaticales ampliando su conocimiento a ideas que viene en los ejemplos, obviamente el modelo transformer debe ser lo suficientemente grande.

Se puede implementar un modelo Transformer desde cero utilizando una biblioteca de DL como TensorFlow o PyTorch. Esto permite tener una comprensión detallada de cómo funcionan estas arquitecturas.

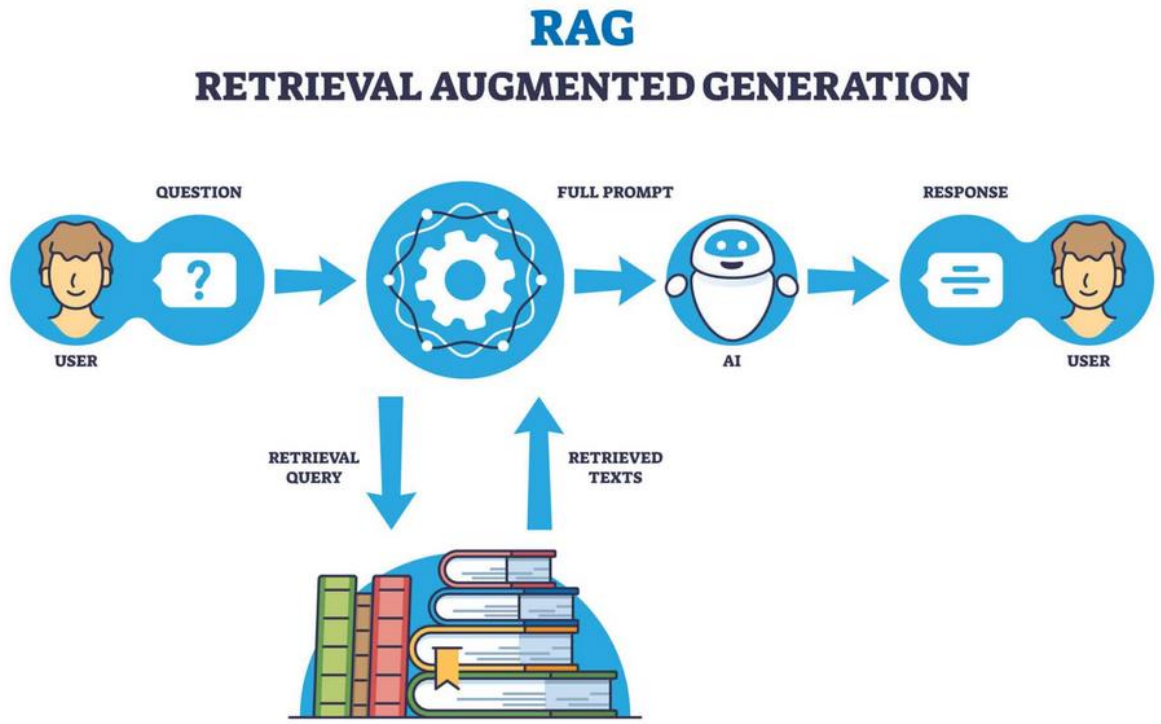
7.7 RAG, Retrieval-Augmented Generation

RAG, Retrieval-Augmented Generation, recuperación por generación aumentada, es una técnica para mejorar la precisión y confiabilidad de los modelos generativos de IA con datos obtenidos de fuentes externas. Es una arquitectura de software que combina las capacidades de los LLMs, que tienen conocimiento general del mundo, con fuentes de información específicas para un negocio como: documentos, bases de datos SQL y aplicaciones internas del negocio. RAG mejora la precisión y la pertinencia de las respuestas de los LLMs.

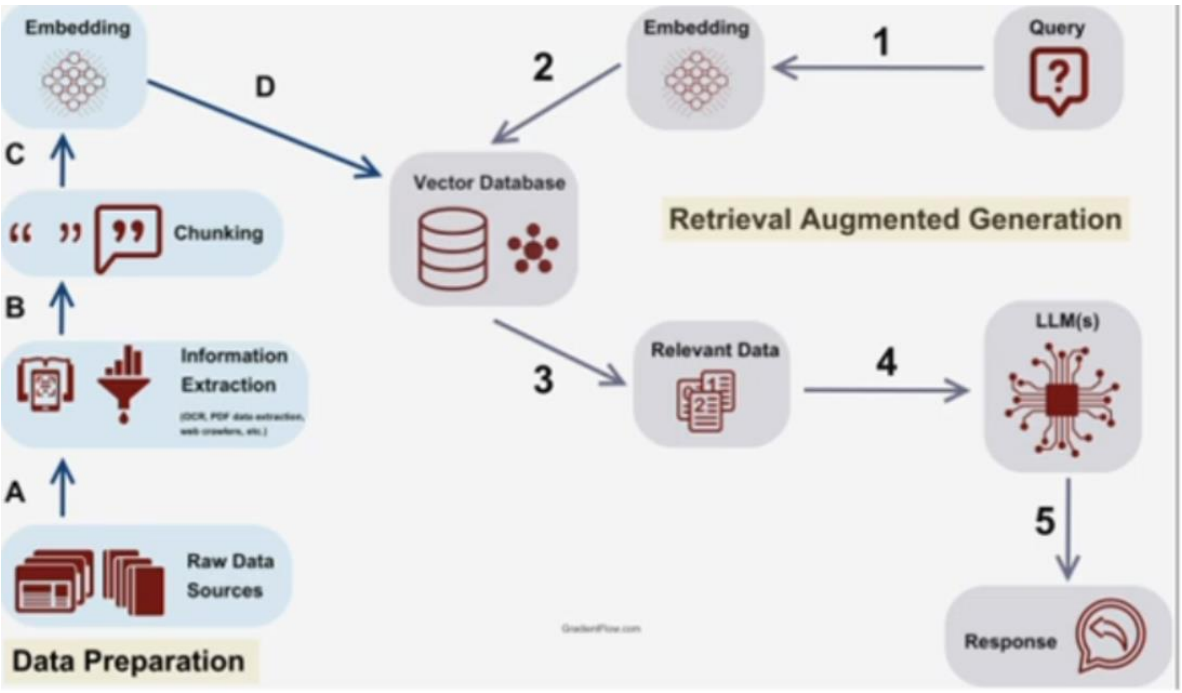
Cuando un MLL no tiene suficiente información, o no tiene conocimiento contextual de un tema, probablemente genere alucinaciones y dé respuestas imprecisas y falsas. El enfoque RAG puede ayudar a superar las limitaciones más significativas de los LLMs, como el conocimiento limitado a los datos de entrenamiento, la falta de contexto pertinente a los datos empresariales y datos no actualizados.



El enfoque RAG, en la medida en que se ha vuelto más popular, se ha hecho evidente que su eficacia depende completamente de la calidad de la búsqueda del sistema base de recuperación. Los LLMs son lo suficientemente inteligentes para comprender y responder preguntas, sin embargo, no se puede utilizar su poder si el sistema base no realiza búsquedas de alta calidad para escanear la gran cantidad de información propietaria.



RAG usa una tecnología de base de datos vectorial para almacenar información actualizada que se recupera usando búsquedas semánticas que se añade a la ventana de contexto del prompt, junto con otra información útil, para permitirle al LLM formular las mejores respuestas actualizadas.



Paso 1. Definición del problema, qué es lo que se quiere consultar sobre el conocimiento específico de una empresa, cual es el tipo de preguntas que va a responder el sistema y de ahí, ver los documentos fuente que se van a usar como base.

Entre las fuentes de datos están: documentos pdf, archivos de texto, páginas web, manuales técnicos, documentos científicos.

Paso 2. Extracción de información. Los documentos, en cualquier formato se deben cargar y convertirse en archivos de texto. Luego se debe hacer una limpieza y eliminar el ruido que puedan tener. Luego, debido a problemas de longitud de contexto, se dividen los documentos de texto en trozos llamados chunks.

Paso 3. Obtención de vectores embedding. Cada símbolo componente del chunk, se convierte en token. Ese token es un número y con ese número se recupera su vector de embedding. Se hace para todos los chunks. Estos vectores embedding se almacenan en una base de datos vectorial para su búsqueda semántica.

Paso 4. Consulta del usuario. Los símbolos del prompt del usuario primero se convierten en tokens y luego en se convierten en vectores embedding.

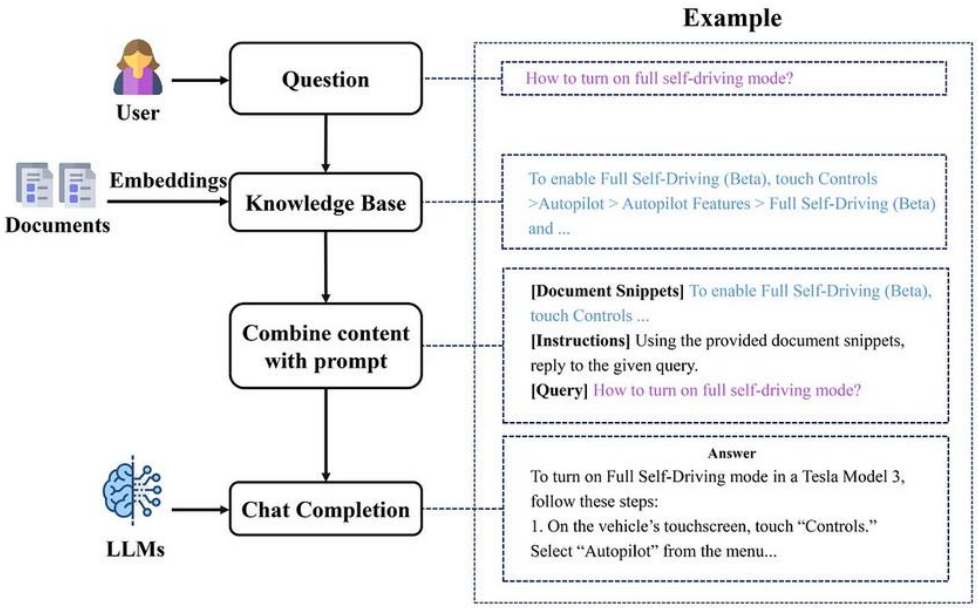
Paso 5. Recuperación de documentos relevantes. Se busca en la base de datos vectorial los chunks más cercanos. Los chunks más cercanos, se recuperan de la base de datos vectorial y se agregan al a los vectores embedding del prompt del modelo generativo.

Paso 8. Respuesta. El LLM, con base en esos vectores embeddings y su estructura transformer genera la respuesta.

7.7.1 Asistentes virtuales con RAG

En general las empresas e instituciones, pueden mejorar su imagen, utilizando asistentes virtuales para ampliar la interacción con los usuarios ofreciendo un servicio de alta calidad las 24 horas al día, siete días a la semana, con información completamente actualizada.

Por ejemplo, si la Universidad tuviera este tipo de servicio, su comunidad estaría enterada de la gran diversidad de ofertas y eventos científico-culturales que desarrolla permanentemente la Universidad, logrando con mayor efectividad sus objetivos misionales.



Flujo en un sistema RAG

7.8. Agentes

Los agentes de IA se definen como programas que pueden realizar tareas en nombre de los usuarios. Sin embargo, donde la situación se vuelve realmente interesante es con el concepto de sistemas multiagente, que implica que varios agentes de IA trabajen juntos para responder consultas o ejecutar flujos de trabajo, imitando la colaboración humana en equipo.

Los sistemas multiagente eficaces consisten en agentes especializados que pueden gestionar dependencias paralelas y ejecutar tareas de forma concurrente o secuencial. Existen plataformas y frameworks completos dedicados a la creación de estos sistemas multiagente, como LangChain y Crew AI.

En concreto, un agente de IA necesita tres cosas para ser útil:

1. Un modelo de IA para usar. Piensa en algo como chatGPT u otro LLM.
2. Memoria para que tenga contexto entre interacciones
3. Herramientas y Conocimientos para que sea capaz de investigar y brindar respuestas precisas

Por ejemplo, si le pregunta al agente cuál es la temperatura actual en Cali, para saber que ropa usar, el agente no podría responder. Necesitaría una conexión a un servicio meteorológico. De manera, que hay que proporcionarle al agente todas las herramientas necesarias para resolver su consulta. En una organización, una herramienta podría ser tu base de conocimiento, si está creando un agente de soporte, o quizá el sistema CRM si se trata de la gestión de cuentas.

Lo que nos lleva al MCP. Hablamos de herramientas, pero ¿cómo se conectan realmente estas herramientas? Sin duda, cada herramienta tiene su propia interfaz, y cada framework de agente tiene sus propias expectativas sobre cómo usarlas.

MCP es un protocolo abierto desarrollado por Anthropic para estandarizar cómo las aplicaciones proporcionan contexto (datos y herramientas) a los LLM. Se compara con un "puerto USB-C para aplicaciones de IA", que ofrece una forma estandarizada de conectar modelos de IA a diferentes fuentes de datos y herramientas.

El objetivo de MCP es abordar el panorama fragmentado actual, donde los desarrolladores crean integraciones personalizadas para que los LLM accedan a datos y herramientas de diferentes maneras. Entonces, en lugar de que cada uno aborde estas integraciones a su manera, existe una forma estandarizada en la que los desarrolladores pueden escribir un servidor MCP, que es básicamente una herramienta que los LLM pueden usar para diferentes tareas.

7.9. Hacia dónde va la IA generativa

Las actuales herramientas de IA Generativa han realizado avances notables pero es bueno anotar que se están en un estado primitivo de desarrollo. Aunque se han demostrado sus impresionantes capacidades, hay varias limitaciones y retos que se deben abordar y que actualmente están en investigación y es plausible que en los siguientes 5 a 10 años, evolucionará a límites sin precedentes, sobrepasando nuestras expectativas actuales.

Las tendencias en IA Generativa incluyen el crecimiento de Agentes de IA, capaces de actuar de forma independiente; la creciente sofisticación de la IA multimodal que procesa texto, imágenes y vídeo; y el auge de la hiper-personalización en servicios y contenido. Las empresas también están observando una adopción generalizada de la automatización de tareas y la aceleración de los flujos de trabajo creativos, a la vez que las consideraciones éticas y la necesidad de regulación cobran mayor relevancia.

- **Agentes de IA**. Un avance importante es la transición hacia Agentes de IA, que se refiere a sistemas de IA que pueden tomar la iniciativa y realizar tareas de forma autónoma sin intervención humana constante.
- **Hiper-personalización.** La IA Generativa facilita a las empresas la creación de servicios, productos y contenido de marketing altamente personalizados, adaptados a las necesidades individuales de los clientes en tiempo real.
- **Generación de contenido creativo.** La IA Generativa continúa acelerando los procesos creativos, desde la escritura y la generación de código hasta el diseño, el marketing y los esfuerzos artísticos.

- **Mejoras en IA conversacional.** Los avances en el procesamiento del lenguaje natural están haciendo que los chatbots y los asistentes virtuales sean más potentes y versátiles, lo que les permite gestionar consultas complejas de manera más eficiente.
- **IA física.** Integra modelos computacionales con el mundo real, permitiendo a máquinas y robots percibir, razonar y actuar en entornos tridimensionales basándose en leyes como la gravedad y la fricción. Tiene el potencial de revolucionar la forma en que concebimos, diseñamos y producimos productos físicos.

Veamos algunas formas que se pueden usar en conjunción con la impresión 3d:

- Diseño generativo de AutoDesk. El software de diseño de Autodesk's Generative utiliza algoritmos de IA para generar diseños optimizados de productos con base en criterios específicos de comportamiento. Ya se ha aplicado en varias industrias como la automovilística y la aeroespacial, para crear componentes ligeros y estructuralmente eficientes.
- Adidas Futurecraft Strung. Adidas utilizó algoritmos de IA Generativa para crear la zapatilla Futurecraft Strung, que presenta una parte superior en forma de enrejado, construida a partir de datos generados por IA. Este enfoque permitió personalizar el ajuste y la sujeción, demostrando el potencial de la IA Generativa para personalizar productos físicos.
- Nervous System's Kinematics Dress. El estudio de diseño Nervous System ha utilizado la IA Generativa para crear el vestido Kinematics, una prenda impresa en 3D que se adapta y ajusta al cuerpo de quien la lleva. El vestido muestra cómo la IA Generativa puede revolucionar la industria de la moda con la producción de prendas complicadas y personalizables.
- Optimización del diseño. La IA generativa, puede asistir en la optimización de diseño de productos para impresión 3D. Introduciendo criterios de diseño y de comportamiento deseado, los algoritmos de IA Generativa pueden generar numerosas iteraciones de diseño. Estos diseños se pueden evaluar en función de factores como la reducción de peso, la integridad estructural, el uso de materiales y la eficiencia de la producción. Luego, se pueden ajustar los resultados y seleccionar para la impresión 3D.
- Geometrías complejas y personalización. La IA generativa permite crear geometrías complejas y personalizadas que se adaptan bien a la impresión 3D. Las técnicas de fabricación tradicionales tienen limitaciones a la hora de producir diseños complicados, pero la impresión 3D permite la realización de tales geometrías. Los algoritmos de IA Generativa pueden generar estructuras únicas y complejas que optimizan el uso de materiales, mejoran la funcionalidad o consiguen cualidades estéticas específicas. Este potencial de personalización permite fabricar productos personalizados adaptados a las necesidades y preferencias individuales.
- Fabricación hibrida. La IA generativa puede utilizarse para optimizar la integración de distintos materiales y procesos de fabricación con enfoques de fabricación híbridos. Por ejemplo, los algoritmos de IA Generativa pueden determinar dónde pueden combinarse técnicas de fabricación tradicionales como el mecanizado CNC o el moldeo por inyección con la impresión 3D para conseguir las propiedades y funcionalidades deseadas de un producto. Esta combinación de diferentes métodos de fabricación permite una mayor flexibilidad, eficiencia y rentabilidad en la producción.

7.10. Impacto social y ecosistema de modelos fundamentales.

(Tomado de On the Opportunities and Risks of Foundation Models, Rishi Bommasani et al, Center for Research on Foundation Models (CRFM) Stanford Institute for Human-Centered Artificial Intelligence (HAI) Stanford University). Pero lo que los hace críticos a modelos fundamentales es que se están integrando rápidamente en implementaciones de sistemas de IA en el mundo real, con consecuencias de gran alcance para las personas.

Pero, ¿cuál es la naturaleza de este impacto social? En este informe abordamos muchos aspectos de esta cuestión: la posible exacerbación de las desigualdades sociales (§5.1: equidad), el impacto económico debido al aumento de sus capacidades (§5.5: economía), el impacto ambiental debido a las mayores demandas de computación y energía (§5.3: medio ambiente), las posibles preocupaciones de ampliar la desinformación (§5.2: uso indebido), las ramificaciones legales debido a las poderosas capacidades generativas (§5.4: legalidad), las cuestiones éticas resultantes de la homogeneización y la economía política más amplia en la que se desarrollan y despliegan los modelos fundamentales (§5.6: ética).

Dada la naturaleza proteica de los modelos fundamentales y sus capacidades no proyectadas, ¿cómo podemos anticipar y abordar responsablemente las consideraciones éticas y sociales que plantean? Un tema recurrente es que es más fácil razonar sobre el impacto social de sistemas específicos implementados para usuarios específicos, que razonar sobre el impacto social de modelos fundamentales, que podrían adaptarse a cualquier número posterior de sistemas no previstos.

Si bien la producción de conocimiento puede desempeñar un papel vital en la configuración del futuro, el impacto social directo se produce a través de la implementación de estos modelos, que se rige por prácticas de propiedad sobre los datos, generalmente privados. A veces, la implementación se realiza a través de nuevos productos, pero a menudo se realiza mediante actualizaciones de productos existentes. Los modelos de investigación a menudo no se prueban exhaustivamente y pueden tener fallas desconocidas; deberían colocarse etiquetas de advertencia en los modelos de investigación que no sean aptos para implementarse. Por otro lado, los modelos fundamentales implementados que realmente afectan la vida de las personas deberían estar sujetos a pruebas y auditorías mucho más rigurosas.

Para comprender mejor la investigación y la implementación de los modelos fundamentales, debemos considerar el ecosistema completo en el que habitan estos modelos, desde la creación de datos hasta la implementación real. Es importante señalar que el modelo fundamental es sólo un componente de un sistema de IA. Simplificando, podemos pensar en el ecosistema de un modelo fundamental en términos de secuencia de etapas, extendiendo las etapas de capacitación y adaptación anteriores. Esta visión del ecosistema nos permite ver que diferentes preguntas sobre los modelos fundamentales en realidad deben responderse con respecto a las diferentes etapas.

(1) Creación de datos: la creación de datos es fundamentalmente un proceso centrado en el ser humano: todos los datos son creados por personas y la mayoría de los datos son, al menos implícitamente, sobre las personas. A veces, los datos son creados por personas para otras personas en forma de correos electrónicos, artículos, fotografías, etc., y a veces son una medición de personas (por ejemplo, datos genómicos) o una medición del entorno en el que viven las personas (por ejemplo, imágenes de satélite). Es importante tener en cuenta que todos los datos tienen un propietario y se crean con un propósito (donde ese propósito puede incluir o no el entrenamiento de un modelo fundamental).

(2) Conservación de datos: luego, los datos se seleccionan en datasets. No existe una distribución natural única de los datos; Incluso el rastreo de Internet más permisivo requiere cierta selección y filtrado posterior. Garantizar la relevancia y la calidad de los datos respetando al mismo tiempo las limitaciones legales y éticas es fundamental, pero desafiante. Si bien esto es reconocido en la industria, está subestimado en la investigación de la IA (§4.6: datos).

(3) Entrenamiento: El entrenamiento de los modelos fundamentales con en estos datasets seleccionados, es la pieza central celebrada en la investigación de la IA, aunque es solo una de muchas etapas.

(4) Adaptación: en el contexto de la investigación de ML, la adaptación consiste en crear un nuevo modelo basado en el modelo fundamental que realiza alguna tarea. Para la implementación, la adaptación consiste en crear un sistema, que requiere potencialmente muchos módulos diferentes, reglas personalizadas y combinación con otras señales. Por ejemplo, un modelo problemático capaz de generar contenido tóxico podría ser tolerable si se toman las precauciones adecuadas posteriormente. La lógica adicional, específica de la aplicación, es crucial para mitigar los daños.

(5) Implementación: el impacto social directo de un sistema de IA se produce cuando se difunde entre las personas. Aunque no quisiéramos implementar modelos fundamentales potencialmente dañinos, entrenados con datos cuestionables, aún podría ser valioso permitirlos en la investigación para avanzar en la comprensión científica, aunque aún hay que tener cuidado. En términos más generales, es una práctica estándar en implementaciones a gran escala realizar lanzamientos graduales, donde la implementación ocurre en una fracción cada vez mayor de usuarios; esto puede mitigar parcialmente cualquier daño potencial.

Si bien las grandes organizaciones pueden poseer todo el proceso, cada etapa podría ser realizada por una organización diferente, por ejemplo, una empresa que se especializa en la creación de modelos fundamentales personalizados para varios dominios que los desarrolladores de aplicaciones puedan utilizar. Piense en que el ecosistema, actúa como modelo. Si bien el impacto social depende de todo el ecosistema, sigue siendo importante poder razonar sobre las implicaciones sociales de un modelo fundamental, dado que el ámbito de muchos investigadores y profesionales se limita a la etapa de entrenamiento.

Esto es difícil porque los modelos fundamentales son objetos intermedios inacabados que pueden adaptarse a muchas aplicaciones posteriores, a veces por parte de una entidad completamente diferente para propósitos imprevistos. Lo que necesitamos son dos cosas: (i) métricas sustitutas para un conjunto representativo de posibles evaluaciones posteriores (§4.4: evaluación), y (ii) un compromiso para documentar estas métricas [Mitchell et al. 2019] similares a las hojas de datos para materiales como metales y plásticos, que pueden adaptarse a muchos casos de uso posteriores.

Caracterizar el posible impacto social posterior de los modelos fundamentales es un desafío y exige una comprensión profunda tanto del ecosistema tecnológico como de la sociedad. No se pueden evaluar plenamente los daños (§5.1: equidad) de un modelo fundamental sin reconocer cómo se implementará, y no se pueden simplemente definir métricas automáticas sin considerar el rico contexto social e histórico.

7.11. Ejercicios

1. Desarrollar un agente de IA, con cualquier chatbot de IA, para el manejo de las tareas del curso.
2. Mejorar el sistema RAG para crear un chat que permita acceder a la información actualizada de la Universidad Nacional, u otra empresa o entidad.
3. Desarrollar un Agentic AI para el manejo personal de sus actividades.
4. Hacer un estudio sobre las posibilidades de la IA Física en nuestro país.

7.12 Bibliografía

J. Devlin et al, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, fjacobdevlin,mingweichang,kentonl,kristoutg@google.com

T. Brown, Language Models are Few-Shot Learners, OpenAI
A. Vaswani, Attention Is All You Need, Google
S. Tedeschi, et al, What's the Meaning of Superhuman Performance in Today's NLU?

https://www.youtube.com/watch?v=4Bdc55j80l8 Transformers

https://www.youtube.com/watch?v=eMlx5fFNoYc Explica los encoders

https://www.youtube.com/watch?v=TQQlZhbC5ps Redes recurrentes y transformer

https://www.google.com/search?client=firefox-b-d&hs=wt&sca_esv=a8e9eb64a8fa43d9&udm=7&fbs=ADc_l-bpk8W4E-qsVlOvbGJcDwpn60DczFdcvPnuv8WQohHLTaMb_WtLz8zQ41bNqiqMK_2tAHCL8fGF2xU6_n5bwc1VQnfR-2i3m-Hxw8AVipIz45kzFjjDCxYaze00BT7L6r9jskn6XEYEdpuNPF3Q09fLgLbNQMsSCb7Clb7u6z4_SxS5TH1Cjo4yXOSLe7ZEfuKq-ee1NA2ljk7_iLjEEAgh_ElWAdmg2Xv_Kl0Ec9Sdf_a98Xc&q=agentic+ai&sa=X&ved=2ahUKEwjvksaD2JqTAxVTQzABHbIyCz0QtKgLegQIFhAB&biw=1482&bih=708&dpr=1.25#fpstate=ive&vld=cid:41659839,vid:C7vqA9TTQ38,st:0 Agentic AI