Anexo A. *Inteligencia artificial*

En el ensayo “Computing Machinery and Intelligence” de Turing (2009), publicado en 1950, el autor propone la famosa pregunta: "¿Pueden las máquinas pensar?". Turing aborda la cuestión de si una máquina puede mostrar comportamientos inteligentes equivalentes a los humanos y, por ende, si puede considerarse que posee inteligencia. En el libro “Inteligencia artificial. Un enfoque moderno” Russell y Norvig (2004) definen la IA como el estudio de los agentes que reciben percepciones del entorno y llevan a cabo las acciones. Cada agente implementa una función la cual estructura las secuencias de las percepciones en acciones.

La inteligencia artificial, un campo dedicado a desarrollar algoritmos con capacidad de pensar y tomar decisiones como seres humanos, se divide en diversas ramas, cada una con sus propios enfoques y metas. El aprendizaje automático se centra en la habilidad de las máquinas para aprender sin programación explícita, utilizando algoritmos que emplean técnicas matemáticas y estadísticas para analizar datos y mejorar el desempeño en tareas específicas. El aprendizaje profundo, una subrama del aprendizaje automático, se basa en redes neuronales artificiales, modelos matemáticos inspirados en el cerebro humano, que pueden aprender a realizar tareas complejas como el reconocimiento de imágenes y el procesamiento del lenguaje natural.

Aprendizaje automático

Se define como un conjunto de métodos que pueden detectar patrones automáticamente en los datos y luego usar esos patrones descubiertos para predecir datos futuros o para tomar otros tipos de decisiones bajo incertidumbre (Murphy, 2012)

Los tipos de aprendizaje automático se dividen en tres: supervisado, no supervisado y por refuerzo. El algoritmo del aprendizaje supervisado incluye un conjunto de datos para el entrenamiento el cual viene con las características de entrada y las soluciones esperadas. Ahora, pasamos al aprendizaje no supervisado, donde solo se nos proporcionan datos de salida, sin ninguna entrada. Por otra parte, el aprendizaje por refuerzo no cuenta con entradas o salidas, este se basa en la idea de que un agente, que interactúa con un entorno, puede aprender a tomar decisiones secuenciales para maximizar una señal de ganancia a lo largo del tiempo.

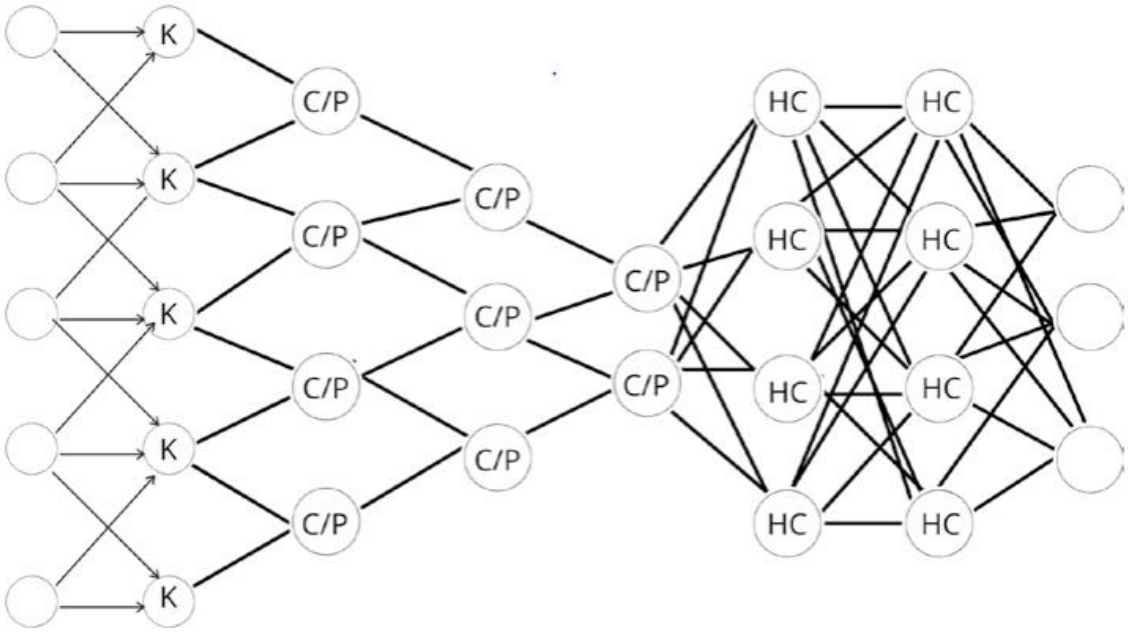
Redes neuronales.

Una red neuronal artificial es un artefacto computacional utilizado para la clasificación y predicción de datos, así como una herramienta para modelado cognitivo. Las redes están compuestas por una colección de unidades computacionales simples e interconectadas, cada una de las cuales puede considerarse un modelo altamente simplificado de una neurona biológica.

Las características biológicas de la neurona real generalmente modeladas por la abstracción incluyen adaptabilidad, computación distribuida en paralelo, no linealidad de la función de entrada a salida y la localidad de la computación de cada neurona. La neurona modelo generalmente suma sus entradas, ponderadas por las fuerzas de conexión entrantes, y produce un único valor de salida.

La función de salida puede ser cualquiera de una variedad de funciones lineales, semi-lineales o no lineales (Seel, 2011)

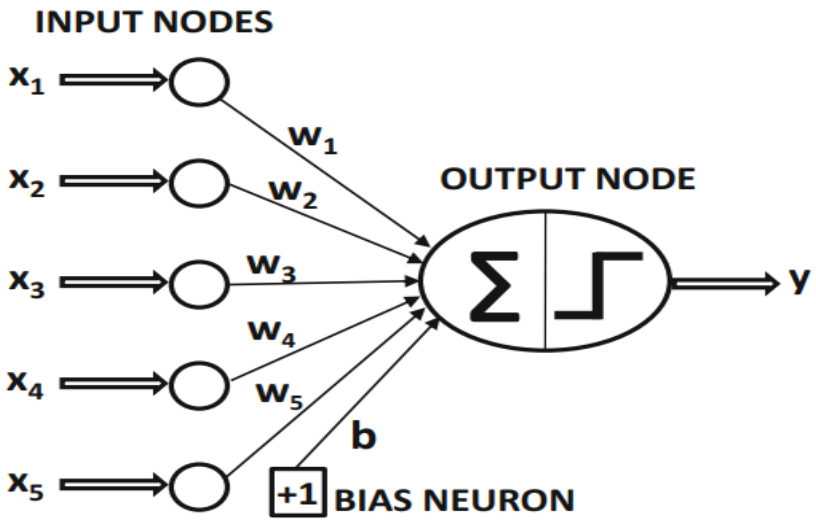
Figura 1. *Ejemplo de red neuronal*



Nota. Ejemplo de Red Neuronal Convolucional (K indica el kernel, C/P indica Convolución o Pooling y HC indica Celda oculta.). Tomado de *Silaparasetty, V. (2020). Neural Networks. In: Deep Learning Projects Using TensorFlow 2. Apress, Berkeley, CA.*

Perceptrón: el perceptrón es un tipo de modelo de red neuronal artificial que se caracteriza por tener una sola capa de neuronas. Fue propuesto por Frank Rosenblatt en 1957 y se utiliza principalmente para problemas de clasificación binaria. La estructura básica de un perceptrón consiste en un conjunto de entradas (x1, x2, ..., xn), cada una multiplicada por un peso correspondiente (w1, w2, ..., wn), y luego sumadas junto con un sesgo (b). El resultado de esta suma se pasa a través de una función de activación, que produce la salida del perceptrón, tal como se muestra en la Figura 2.

Figura 2. *Perceptrón*



Nota. Perceptrón con Sesgo. Tomado de *Charu C. Aggarwal. Neural Networks and Deep Learning. 1.a ed. NY: Springer International Publishing AG, 2018. 512 págs.*

Matemáticamente, la salida (y) de un perceptrón se puede expresar como se muestra en la ecuación 1.

$$y = f(w1 \cdot x1 + w2 \cdot x2 + \ldots + wn \cdot xn + b) \qquad (1)$$

La función de activación (f) es un elemento fundamental del perceptrón. Es la que determina el comportamiento del perceptrón y la clase a la que se asigna la entrada. La función de activación más común es la función umbral (threshold), que produce una salida binaria (0 o 1) basada en si el resultado de la suma es mayor o menor que un cierto umbral. El perceptrón es un modelo relativamente simple y tiene algunas limitaciones. En particular, no puede aprender patrones no lineales. Esto significa que no puede abordar problemas complejos, como la clasificación de imágenes o la traducción automática. A pesar de sus limitaciones, el perceptrón es un modelo importante en el desarrollo de las redes neuronales artificiales. Fue el primer modelo de red neuronal que se propuso y sentó las bases para

arquitecturas más avanzadas, como las redes neuronales multicapa. El perceptrón es un modelo de red neuronal artificial simple pero fundamental. Es un modelo importante en el desarrollo de las redes neuronales artificiales y se utiliza para una variedad de aplicaciones, como la clasificación binaria.

En muchos escenarios, existe una parte invariante de la predicción, que se conoce como sesgo. Por ejemplo, se considera un escenario en el que las variables de características están centradas en la media, pero la media de la predicción de clase binaria de {-1, +1} no es 0. Esto tenderá a ocurrir en situaciones en las que la distribución de clases binarias está muy desequilibrada. En tal caso, el enfoque mencionado anteriormente no es suficiente para la predicción. Es necesario incorporar una variable de sesgo adicional b que capture esta parte invariante de la predicción tal como se muestra en la ecuación 2.

$$\hat{y} = sign\{W \cdot X + b\} = sign\{\sum_{j=1}^{d} w_j \cdot x_j + b\} \qquad (2)$$

Backpropagation El algoritmo de backpropagation es una aplicación directa de la programación dinámica. Se compone de dos fases principales, denominadas fase de forward y fase de backward, respectivamente. La fase de forward es necesaria para calcular los valores de salida y las derivadas locales en varios nodos, mientras que la fase de backward es necesaria para acumular los productos de estos valores locales a lo largo de todos los caminos desde el nodo hasta la salida (Aggarwal, 2018). La fase de backward es un proceso de aprendizaje que utiliza la regla de la cadena del cálculo diferencial para calcular el gradiente de la función de pérdida con respecto a los diferentes pesos de una red neuronal. Estos gradientes se utilizan para actualizar los pesos de la red, lo que permite mejorar la precisión de la red en la tarea de aprendizaje. Este cálculo de gradiente podremos encontrarlo en la ecuación 3.

$$x_{i+1} = xi - \alpha \frac{df}{dx}$$

Metafóricamente α (o tasa de aprendizaje) representa la velocidad a la que un modelo de aprendizaje automático "aprende". Al establecer una tasa de aprendizaje, hay un equilibrio entre la rapidez de convergencia y la sobreoscilación. Mientras que la dirección de descenso generalmente se determina a partir del gradiente de la función de pérdida, la tasa de aprendizaje determina cuán grande es el paso en esa dirección. Una tasa de aprendizaje demasiado alta hará que el aprendizaje salte sobre mínimos, pero una tasa de aprendizaje demasiado baja llevará demasiado tiempo converger o quedará atrapada en un mínimo local indeseado (Avgoustis, 2021)

Optimizadores se utilizan para ajustar los pesos de la red durante el proceso de entrenamiento con el objetivo de minimizar la función de pérdida. Algunos de los optimizadores comunes incluyen el Descenso del Gradiente Estocástico (SGD), Momentum, Adagrad, RMSprop, y Adam. La idea principal detrás del Momentum es agregar un término que tenga en cuenta la dirección y la velocidad de la iteración anterior para ajustar los pesos durante el entrenamiento En el caso del SGD es el optimizador más básico. Actualiza los pesos en la dirección opuesta al gradiente de la función de pérdida con respecto a los pesos, multiplicado por la tasa de aprendizaje.

El Adagrad ajusta la tasa de aprendizaje para cada parámetro en función del histórico de gradientes de ese parámetro. Los parámetros que tienen grandes gradientes obtendrán tasas de aprendizaje más pequeñas, y viceversa. Similar al anterior, el RMSprop utiliza una ventana móvil del cuadrado medio de los gradientes en lugar de su suma acumulativa. Esto ayuda a mitigar el problema de que la tasa de aprendizaje se vuelva demasiado pequeña en etapas posteriores del entrenamiento. Por último, tenemos al optimizador Adam que tiene

ideas de RMSprop y Momentum. Utiliza tanto los momentos del primer orden (media móvil exponencial del gradiente) como del segundo orden (media móvil exponencial del cuadrado del gradiente) para adaptar la tasa de aprendizaje de cada parámetro individualmente (Heaton, 2018).

Funciones de activación como se mencionó antes una red necesita propagar información en ambas direcciones para actualizar los pesos. Dentro de este ejercicio se pueden presentar los siguientes inconvenientes: Gradientes que se desvanecen: La fuerza de la señal disminuye a medida que fluye a través de las capas, dificultando el aprendizaje. Gradientes que explotan: La fuerza de la señal se amplifica de manera incontrolable, saturando la activación y perturbando el entrenamiento. Para abordar estos problemas, los autores Glorot y Bengio proponen (Aurélien, 2019) mantener una varianza igual entre las capas: Varianza de salida: La varianza de la salida de cada capa debe ser igual a la varianza de su entrada. Varianza de gradiente: La varianza de los gradientes antes y después de una capa debe ser aproximadamente igual papel de las funciones de activación:

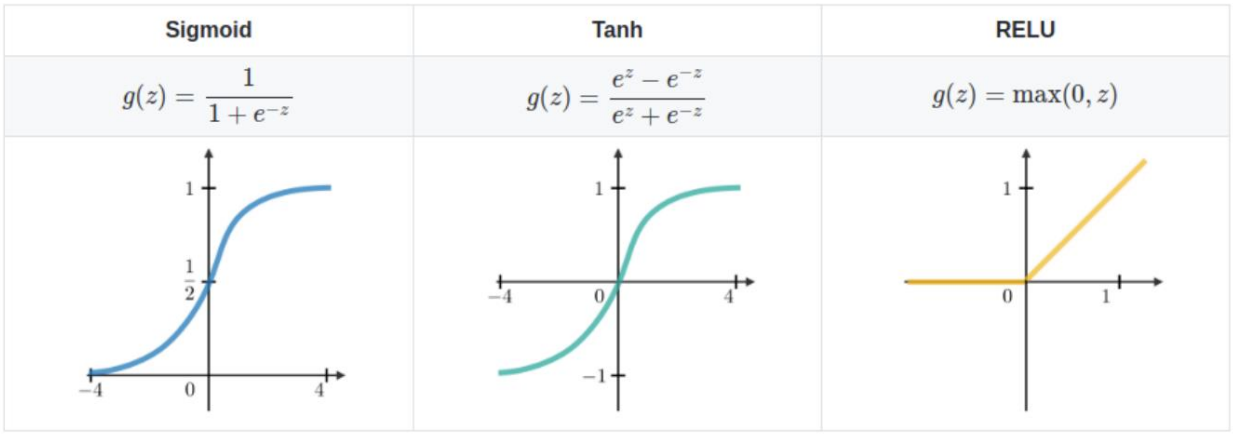
No linealidad: Las funciones de activación introducen no linealidad en la red, esencial para tareas complejas. Los modelos lineales no pueden aprender tales tareas de manera efectiva.

Flujo de señal controlado: Las funciones de activación específicas pueden ayudar a regular el flujo de señal y abordar los gradientes que se desvanecen o explotan. Por ejemplo: Sigmoide/Tanh: Aplastan los valores entre 0 y 1, limitando la amplitud de la señal pero sufriendo de gradientes que se desvanecen.

ReLU: Unidad lineal rectificada, permite que los gradientes fluyan libremente para los valores positivos, lo que potencialmente puede conducir a explosiones.

Variaciones como Leaky ReLU/ELU: Introducen pequeñas pendientes no nulas para los valores negativos, mitigando los gradientes que se desvanecen al tiempo que mantienen los beneficios de ReLU.

Figura 3. *Funciones de activación*



Capas convolucionales (Padding-Kernel-stride) Las capas convolucionales son un componente fundamental en las redes neuronales convolucionales (CNNs), las cuales son un tipo de red neuronal artificial altamente efectiva para el procesamiento de imágenes y datos espaciales. Estas capas se encargan de extraer características y patrones relevantes de la entrada, permitiendo a la red aprender relaciones complejas y realizar tareas como la clasificación de imágenes, la detección de objetos y el reconocimiento facial.

A continuación, se describe en detalle cada uno de los elementos que conforman una capa convolucional:

El relleno (*padding*) se refiere a la adición de elementos ficticios (generalmente ceros) alrededor de la entrada antes de la convolución. Esto permite controlar el tamaño de la salida y evitar que la imagen se reduzca drásticamente durante el proceso de convolución.

El filtro (*kernel*), también conocido como filtro, es una matriz de pesos que se desliza sobre la entrada para realizar la convolución. El tamaño del kernel determina el área de la entrada que se considera en cada paso. Los valores del kernel representan la importancia relativa de cada posición dentro del área de convolución.

El desplazamiento o paso (*stride*) controla la frecuencia con la que el kernel se desliza sobre la entrada. Un stride mayor implica que el kernel se desplaza más rápido, generando una salida de menor tamaño. Un stride de 1 significa que el kernel se mueve una posición en cada paso, mientras que un stride de 2 implica que se mueve dos posiciones, y así sucesivamente.

Agrupación máxima (MaxPooling) es una operación de downsampling (submuestreo) comúnmente utilizada en redes neuronales convolucionales (CNNs) para reducir la dimensionalidad espacial de una entrada mientras retiene las características más relevantes. Esta técnica consiste en dividir la entrada en regiones o ventanas de tamaño fijo y, para cada ventana, seleccionar el valor máximo como el nuevo elemento de la salida. Esto permite que la red mantenga la información más importante (como bordes o patrones destacados) y, al mismo tiempo, reduce la carga computacional.

A continuación, se describe en detalle el funcionamiento del max pooling:

División de la entrada en ventanas: La entrada, que generalmente es una matriz tridimensional (altura, ancho, canales), se divide en regiones o ventanas de tamaño fijo. Por ejemplo, una ventana de 2x2 significa que se divide la entrada en regiones de 2 filas por 2 columnas.

Cálculo del valor máximo: Para cada ventana, se calcula el valor máximo entre todos los elementos dentro de la ventana. Este valor máximo se convierte en el nuevo elemento de la salida en la posición correspondiente a la ventana.

Reducción de la dimensionalidad espacial: La salida del MaxPooling tiene una dimensionalidad espacial reducida en comparación con la entrada. Por ejemplo, si la entrada tiene un tamaño de 10x10 y se utiliza una ventana de 2x2 con un stride de 2 (paso con el que se mueve la ventana), la salida tendrá un tamaño de 5x5.

Aunque MaxPooling no es una técnica de regularización por sí misma, contribuye a evitar el sobreajuste al reducir la cantidad de parámetros que se propagan a las capas siguientes, lo cual simplifica la estructura del modelo y mejora su capacidad de generalización.

Capa Lineal: Las capas lineales, también conocidas como capas completamente conectadas o densas, son un componente fundamental en las redes neuronales artificiales (RNA). Estas capas se encuentran al final de la mayoría de las arquitecturas neuronales y se encargan de transformar la salida de la capa anterior en una única salida final. La función principal de la capa lineal es combinar las activaciones de las neuronas de la capa anterior para generar un resultado predictivo o de clasificación.

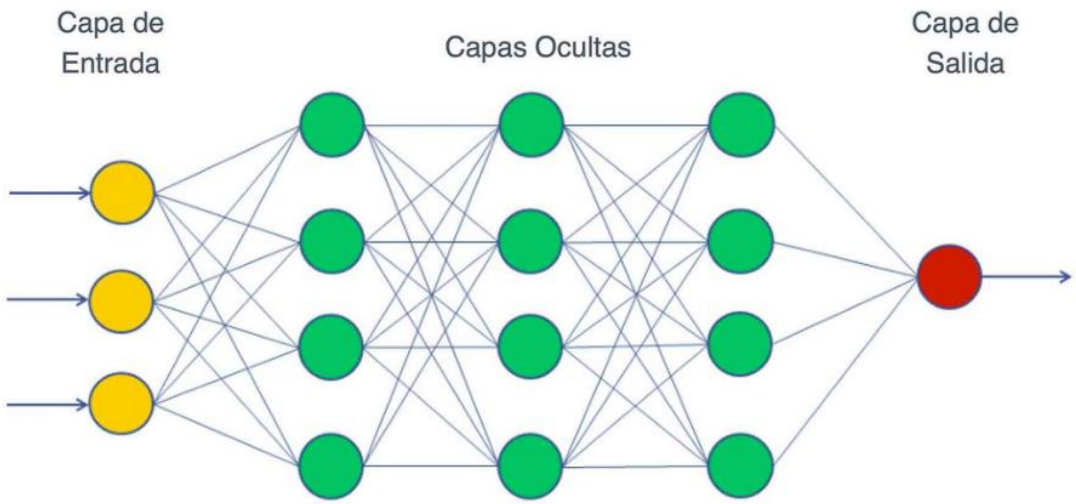
Dropout, también conocido como "abandono" en español, es una técnica de regularización utilizada en redes neuronales artificiales (RNA) para prevenir el sobreajuste. El sobreajuste se produce cuando una red neuronal aprende las características específicas del conjunto de entrenamiento con tanta precisión que no puede generalizar bien a nuevos datos no vistos. El Dropout funciona desactivando aleatoriamente un subconjunto de neuronas durante el entrenamiento, lo que obliga a la red a aprender representaciones más robustas y menos dependientes de neuronas específicas.

Redes multicapa son sistemas computacionales que constan de más de una capa, a diferencia de perceptrones simples. Mientras que en un perceptrón la capa de salida realiza todos los cálculos visibles al usuario, en las redes multicapa existen capas intermedias llamadas capas ocultas, donde los cálculos no son directamente visibles. La arquitectura

específica de estas redes, conocida como feed-forward, implica que la información fluye en una dirección, desde la capa de entrada hasta la capa de salida, sin retroalimentación. Las redes neuronales multicapa estructuran la información en capas intermedias para realizar cálculos no visibles al usuario, con un flujo unidireccional de información desde la entrada hasta la salida y una conexión definida entre capas. La función de pérdida en la capa de salida es clave para la optimización del modelo.

Redes neuronales profundas una red neuronal profunda es una red multicapa. Sin embargo, estas redes profundas pueden tener varias capas ocultas con millones de neuronas conectadas como se muestra en la figura 4.

Figura 4. *Ejemplo de red neuronal*



Función de pérdida de entropía cruzada (CrossEntropyloss): es ampliamente utilizada en problemas de clasificación de imágenes porque mide la diferencia entre la distribución de probabilidad predicha por el modelo y la distribución real de las etiquetas. Es ideal para tareas de clasificación con múltiples clases, ya que combina el LogSoftmax y la Negative Log Likelihood (**LogSoftmax** convierte las salidas de una red neuronal en

probabilidades logarítmicas, asegurando que la suma de las probabilidades sea 1. La **Negative Log Likelihood (NLL)** mide la diferencia entre las probabilidades logarítmicas predichas y las etiquetas reales, penalizando más cuando la probabilidad de la clase correcta es baja) en una sola operación, transformando las salidas del modelo en probabilidades y comparándolas con la etiqueta real. CrossEntropyLoss penaliza fuertemente predicciones incorrectas que tienen alta probabilidad y, al ajustar los pesos de la red durante el entrenamiento, busca que las probabilidades de las clases correctas se acerquen a 1, mejorando la precisión del modelo. Su uso facilita la convergencia del modelo durante el entrenamiento, permitiendo que la red se enfoque en minimizar el error de clasificación