Apuntes Inteligencia Artificial, Clase 02 de Octubre

Javier Alonso Rojas Rojas
Escuela de Ingeniería en Computación
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
javrojas@estudiantec.cr

***Abstract*—Estos apuntes reflejan lo conversado en la clase del 02 de octubre donde se mencionaron temas como. El funcionamiento de los agentes basados en modelos de lenguaje de gran escala (LLM) y su papel en la inteligencia artificial moderna. Se mencionaron las principales herramientas y frameworks para la creación de agentes, junto con las diferencias entre sistemas de un solo agente y multiagente. Además, se examina el caso de Sora de OpenAI como ejemplo de modelo multimodal de generación de video y audio, considerando también los retos éticos asociados. Finalmente, se incluye un repaso de los fundamentos de las redes neuronales, sus funciones de activación y el proceso de entrenamiento mediante backpropagation, como base conceptual de los LLM actuales.**

I. Introduction

La inteligencia artificial (IA) ha avanzado rápidamente gracias a los modelos de lenguaje de gran escala (LLMs) y al desarrollo de agentes inteligentes capaces de actuar y razonar en distintos contextos. Estos sistemas han transformado la generación de texto en un proceso de planificación y ejecución más complejo, permitiendo la creación de agentes autónomos que integran herramientas y colaboran entre sí.

Los apuntes abordan los fundamentos y la arquitectura de los agentes basados en LLM, distinguiendo entre sistemas individuales y multiagente, y destacando el papel del *Chain of Thought* (CoT) como mecanismo clave para el razonamiento estructurado. Además, se incluye el caso de estudio **Sora** de OpenAI y un repaso de las bases neuronales del aprendizaje profundo, como las funciones de activación y el entrenamiento por *backpropagation*.

II. Mención de lectura de agentes

Se hizo un repaso general de la lectura "From Language to Action: A Review of Large Language Models as Autonomous Agents and Tool Users". Explicó que lo fundamental para el próximo quiz del martes es comprender lo esencial: los modelos de lenguaje (LLMs) han pasado de ser simples generadores de texto a actuar como agentes autónomos con capacidad para razonar, planificar, usar memoria e interactuar con herramientas externas. La lectura también distingue entre sistemas de un solo agente y sistemas multiagente, en los que varios modelos cooperan para resolver tareas más complejas. Además, se analizan sus aplicaciones en áreas como la investigación, la programación, la salud, la robótica y las simulaciones, junto con los principales desafíos que enfrentan: la memoria limitada, la seguridad, la ética y la necesidad de mejores métodos de evaluación.

Fig. 1. Sora 2

III. Sora 2 by OpenAI

Sora 2 es la nueva versión del modelo multimodal de OpenAI, capaz de generar video y audio sincronizados a partir de texto. Presenta notables mejoras en realismo físico, coherencia visual y control creativo. Su función de *cameos* permite insertar la imagen y voz del usuario, bajo consentimiento, ampliando las posibilidades narrativas y expresivas. Además, han desarrollado un tipo de red social donde se pueden compartir los videos creados con el modelo.

El modelo ofrece mayor precisión en iluminación, movimiento y sonido, además de opciones de control estilístico mediante *steerability*. OpenAI ha incorporado medidas éticas para evitar la reproducción de personas reales o la generación de contenido sensible, lanzándolo de forma gradual mediante la *Sora app* y futuras APIs. Sora 2 representa un avance significativo en la generación audiovisual con IA, aunque plantea retos importantes en materia de privacidad y autenticidad digital.

IV. Repaso de Redes Neuronales

A. El Perceptrón y su evolución

El perceptrón puede entenderse de forma similar a una regresión logística, aunque se diferencia en la función de pérdida que utiliza. Durante la historia de la inteligencia artificial surgió el llamado *"invierno de la IA"*, en parte debido al problema del XOR, ya que este no podía ser representado adecuadamente por un modelo lineal ni por un perceptrón simple.

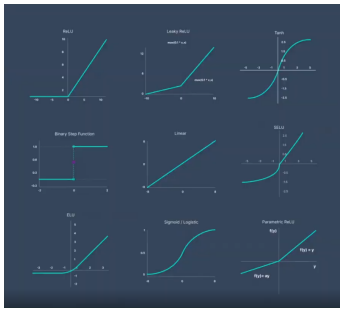

Fig. 2. Funciones de activación

B. El problema del XOR

El principal inconveniente del perceptrón simple es que el problema XOR no es linealmente separable, por lo que este modelo no puede ofrecer una solución adecuada. Esto dio origen a las redes neuronales multicapa (MLP), capaces de resolver problemas no lineales y ampliar significativamente el rango de aplicaciones posibles.

C. Inspiración biológica

Las redes neuronales artificiales se inspiran en el funcionamiento del cerebro humano. Cada neurona recibe señales a través de sus dendritas (entradas), las procesa en el núcleo mediante una combinación lineal, y decide si transmite o no la señal según una función de activación.

D. Funciones de activación

Las funciones de activación introducen no linealidad en el modelo, permitiendo que la red aprenda relaciones complejas:

- **ReLU:** $g(x) = \max(0, x)$; eficiente, pero puede generar "neuronas muertas" cuando el gradiente es cero.
- **Leaky ReLU:** introduce una pequeña pendiente en la parte negativa para evitar neuronas inactivas.
- **Tanh:** produce salidas en el rango $(-1, 1)$, útil para manejar valores positivos y negativos.
- **Sigmoide:** transforma la entrada en valores entre 0 y 1, común en tareas de clasificación binaria.

E. Perceptrón Multicapa (MLP)

El *Multilayer Perceptron* (MLP) extiende el perceptrón simple añadiendo capas ocultas que permiten resolver problemas no lineales. Su estructura general incluye:

- **Capa de entrada:** recibe los datos originales X_i.
- **Capas ocultas:** realizan transformaciones y cálculos internos.
- **Capa de salida:** entrega el resultado final, cuyo tamaño depende del tipo de problema.

El entrenamiento se realiza mediante *backpropagation*, que calcula el error del modelo y ajusta los pesos utilizando descenso de gradiente.

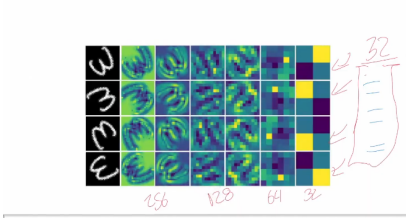

Fig. 3. Comportamiento Jerárquico

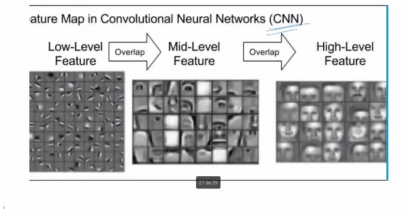

Fig. 4. Funcionamiento de las CNN

Cada capa se calcula de la siguiente forma:

$$h^{(0)} = \sigma(XW_0 + b_0) \tag{1}$$

$$h^{(1)} = \sigma(h^{(0)}W_1 + b_1) \tag{2}$$

$$h^{(n)} = g(h^{(n-1)}W_n + b_n) \tag{3}$$

F. Capas de salida y distribución

Las salidas pueden ser categóricas o continuas:

- En clasificación, se usa *softmax* como función de salida.
- En regresión, se emplea una función lineal.

En todos los casos, la activación final $g(x)$ debe ser no lineal para permitir un aprendizaje más expresivo.

G. Maldición de la dimensionalidad

Cuando se trabaja con datos de muchas variables, los puntos se dispersan en un espacio de alta dimensión, reduciendo su densidad y dificultando el hallazgo de patrones significativos.

H. Comportamiento jerárquico

Como vemos en la Figura 3, las redes neuronales aprenden de forma jerárquica, combinando funciones simples para formar otras más complejas. Esto permite construir representaciones compactas y eficientes, en las que un número reducido de pesos puede modelar funciones avanzadas.

I. CNN

En las redes convolucionales (CNN), las primeras capas detectan bordes o patrones básicos, las intermedias aprenden estructuras más definidas y las últimas capas reconocen objetos completos, como rostros o figuras, esto representado en la Figura 4.

J. Representaciones vectoriales

En el procesamiento de lenguaje natural (NLP), las palabras se representan como vectores en un espacio de alta dimensión, de modo que las palabras con significados o funciones similares se ubican próximas entre sí en dicho espacio.

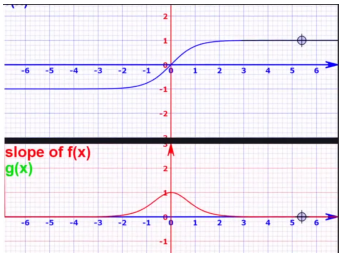

Fig. 5. Tangente hiperbólica

V. Funciones de Activación

Las funciones de activación son un componente esencial en las redes neuronales, ya que permiten introducir la no linealidad necesaria para modelar relaciones complejas entre los datos. A continuación, se describen las funciones más relevantes junto con sus principales características matemáticas.

A. Lineal

La función lineal se define como:

$$f(x) = ax$$

Su derivada es constante ($f'(x) = a$), por lo que el modelo no puede aprovechar el descenso del gradiente para aprender patrones complejos. Debido a su carácter estrictamente lineal, no introduce capacidad de generalización ni no linealidad en la red.

B. Sigmoide

La función sigmoide produce salidas entre 0 y 1, es siempre positiva, acotada y estrictamente creciente:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

A pesar de su utilidad inicial, presenta el problema del *vanishing gradient*: la derivada tiende a cero en los extremos de la función, lo que hace que el aprendizaje sea lento o incluso se detenga en redes profundas.

C. Tangente Hiperbólica (Tanh)

La función tangente hiperbólica tiene un rango de salida entre $(-1, 1)$ como vemos en la Figura 5 y su forma es similar a la sigmoide, pero centrada en el origen:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Esto permite representar tanto valores positivos como negativos, lo que facilita la convergencia del modelo. Sin embargo, al igual que la sigmoide, también sufre del problema del gradiente desvanecido en los extremos.

D. Parametric ReLU (PReLU)

La función **Parametric ReLU (PReLU)** es una variante de la función ReLU tradicional, como se muestra en la figura **??**. A diferencia de la ReLU estándar, esta introduce un parámetro α que se aprende durante el entrenamiento y controla la pendiente en la región negativa. De esta manera, el modelo puede ajustar automáticamente el grado de "fuga" en los valores menores que cero, evitando el problema de las *neuronas muertas*.

Su definición matemática es la siguiente:

$$g(x) = \begin{cases} \alpha x, & \text{si } x < 0 \\ x, & \text{si } x \geq 0 \end{cases}$$

La derivada correspondiente es:

$$\frac{dg(x)}{dx} = \begin{cases} \alpha, & \text{si } x < 0 \\ 1, & \text{si } x \geq 0 \end{cases}$$

El parámetro α se entrena junto con el resto de los pesos de la red, lo que otorga al modelo mayor flexibilidad y capacidad de adaptación frente a distintas distribuciones de datos. Por esta razón, la PReLU suele ofrecer un mejor desempeño en arquitecturas profundas donde la ReLU estándar podría perder gradiente.

E. Softmax

La función Softmax transforma las salidas de la capa final en una distribución de probabilidad, como vemos en la figura 6. Su expresión se define como:

$$\sigma(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$$

donde cada valor x_j se denomina *logit*. Esta función se utiliza principalmente en problemas de clasificación multiclase, ya que garantiza que todas las salidas sean positivas y sumen 1.

- El uso de e^x asegura una función estrictamente creciente y evita valores negativos.
- Se emplea junto con la función de pérdida **Cross-Entropy Loss**, también llamada *Log-Loss*.

La pérdida se define como:

$$L = -\log P(Y = y_i | X = x_i)$$

y, en el caso multiclase:

$$L = -\log \frac{e^{s_k}}{\sum_{j=1}^{C} e^{s_j}}$$

F. Selección de la función de activación

La elección de la función de activación depende del tipo de problema y la arquitectura de la red. Las funciones **Sigmoid** y **Tanh** tienden a sufrir el problema del gradiente desvanecido, por lo que no son recomendadas para redes profundas. En la práctica, se suele comenzar con la función **ReLU** por su eficiencia computacional y buen rendimiento en modelos de *Deep Learning*. Si esta presenta problemas (por ejemplo, neuronas muertas), se pueden utilizar variantes como **Leaky ReLU** o **Parametric ReLU**, que permiten mantener un flujo de gradiente estable incluso en valores negativos.

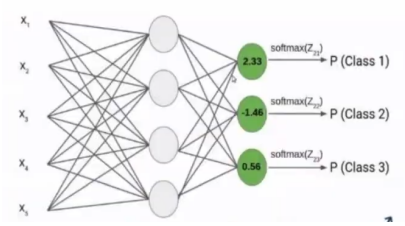

Fig. 6. Uso de Softmax

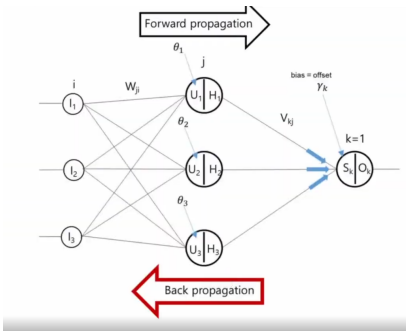

Fig. 7. Forward Propagation y Back Propagation

VI. Backpropagation

El algoritmo de **backpropagation** permite calcular cuánto contribuye cada peso al error final de la red, actualizando los parámetros en la dirección opuesta a la propagación hacia adelante. Este proceso es esencial para que la red neuronal aprenda y mejore su desempeño durante el entrenamiento. Esto representado en la figura 7.

A. Forward Propagation

Consiste en calcular la salida de la red, enviando los datos desde la capa de entrada hacia las capas ocultas hasta obtener la salida final.

B. Backpropagation

implica propagar el error desde la capa de salida hacia las capas anteriores, calculando las derivadas parciales respecto a los pesos y sesgos para ajustar los parámetros del modelo.

C. Optimización del grafo computacional

Consideremos una red neuronal como la de la figura 8, donde cada capa contiene una única neurona y la función de activación utilizada es la sigmoide. El cálculo se puede dividir en las siguientes partes:

- **Función de pérdida (MSE):**

$$L_i = (a^{(L)} - y_i)^2$$

donde $a^{(L)}$ es la salida de la red y y_i el valor esperado.

- **Entrada:**

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

- **Salida:**

$$a^{(L)} = g(z^{(L)})$$

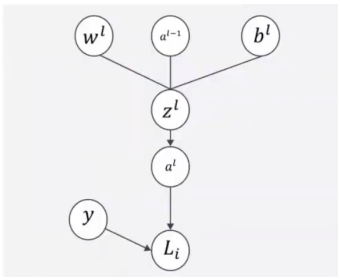

Fig. 8. Red neuronal simple

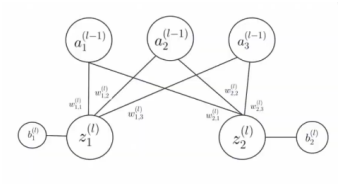

Fig. 9. Red neuronal mas compleja

donde g representa la función de activación.

Los parámetros $w^{(L)}$ y $b^{(L)}$ se actualizan utilizando la regla de la cadena, derivando el error con respecto a cada parámetro y aprovechando la salida de la capa anterior.

D. Vector gradiente

El **vector gradiente** está formado por todas las derivadas parciales de los parámetros (pesos y sesgos) de la red. Durante el cálculo del gradiente se identifican operaciones repetidas, lo que permite optimizar los cálculos en el algoritmo de back-propagation mediante reutilización de resultados intermedios (*caché*).

E. Redes con múltiples neuronas

En redes con mayor dimensionalidad como la de la figura 9, se introducen notaciones adicionales:

- **Superíndice:** indica la capa. Ejemplo: $a^{(l)}$ representa la activación en la capa l.
- **Subíndice:** indica la neurona dentro de una capa. Ejemplo: $a_j^{(l)}$ es la j-ésima neurona en la capa l.
- **Pesos:** se representan como $w_{j,k}^{(l)}$, que conecta la neurona $a_k^{(l-1)}$ con $a_j^{(l)}$, aca j seria el destino y k el origen.

Cada neurona de la capa l recibe entradas desde todas las neuronas de la capa anterior $(l-1)$, siguiendo los pasos:

- **Preactivación:**

$$z_j^{(l)} = b_j^{(l)} + \sum_{k=1}^{n_{l-1}} w_{j,k}^{(l)} a_k^{(l-1)}$$

- **Activación:**

$$a_j^{(l)} = g(z_j^{(l)})$$

Para obtener la activación de una neurona destino, se calculan las contribuciones de todas las neuronas de la capa anterior, multiplicando los pesos de conexión correspondientes por la activación de cada neurona origen. Posteriormente, se

suman estos productos junto con el sesgo asociado, repitiendo el proceso para cada neurona de la capa.

F. Función de pérdida global

La función de pérdida global se obtiene sumando las diferencias entre la salida de cada neurona en la capa de activación l y su valor esperado y_j:

$$L_i = \sum_{j=1}^{n_l} (a_j^{(l)} - y_j)^2$$

G. Aplicación de la regla de la cadena

Dado que las funciones L_i, $z_j^{(l)}$ y $a_j^{(l)}$ están encadenadas, es necesario aplicar la regla de la cadena para derivar cada peso $w_{j,k}^{(l)}$ y sesgo $b_j^{(l)}$. Solo la derivada $\frac{\partial z_j^{(l)}}{\partial w_{j,k}^{(l)}}$ cambia con cada peso actualizado, mientras que las demás se mantienen constantes dentro de la capa.

Las derivadas parciales relevantes son:

$$\frac{\partial L_i}{\partial a_j^{(l)}} = 2(a_j^{(l)} - y_j)$$

$$\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = g(z_j^{(l)})(1 - g(z_j^{(l)}))$$

$$\frac{\partial z_j^{(l)}}{\partial w_{j,k}^{(l)}} = a_k^{(l-1)}$$

A partir de estas derivadas, los pesos y sesgos se actualizan siguiendo el descenso del gradiente:

$$w_{j,k}^{(l)} \leftarrow w_{j,k}^{(l)} - \eta \frac{\partial L_i}{\partial w_{j,k}^{(l)}}, \quad b_j^{(l)} \leftarrow b_j^{(l)} - \eta \frac{\partial L_i}{\partial b_j^{(l)}}$$

donde η representa la tasa de aprendizaje.

En redes más profundas, al extender el cálculo hacia capas anteriores $(l-1)$, el número de parámetros y combinaciones a derivar aumenta considerablemente, incrementando la complejidad computacional del algoritmo.