Apuntes Semana 8 - 10/04/2025

Jose Pablo Barquero Díaz - 2022119938 [1]

***Abstract*— Este documento presenta un análisis detallado de las funciones de activación más utilizadas, tales como Sigmoide, TanH, ReLU y sus variantes. Asimismo, se profundiza en el proceso de Backpropagation, fundamental para el entrenamiento de redes mediante descenso por gradiente.**

I. Noticias de la Semana

A. Llama 4

Meta comparte sus primeros tres modelos en la familia Llama 4.

- Llama 4 Scout: Un modelo de parámetros activos de 17 mil millones con 16 expertos, es el mejor modelo multimodal del mundo en su clase y es más potente que todos los modelos Llama de la generación anterior, al tiempo que se adapta a una sola GPU NVIDIA H100.
- Llama 4 Maverick: Un modelo de parámetros activos de 17 mil millones con 128 expertos, es el mejor modelo multimodal de su clase, superando a GPT-4o y Gemini 2.0 Flash en una amplia gama de puntos de referencia ampliamente informados
- Llama 4 Behemoth: Un modelo de parámetros activos de 288 mil millones con 16 expertos que es nuestro LLM más potente y entre los LLM más inteligentes del mundo, Llama 4 Behemoth supera a GPT-4.5, Claude Sonnet 3.7 y Gemini 2.0 Pro en varios puntos de referencia STEM.

`https://ai.meta.com/blog/llama-4-multimodal-intelligence/`

B. Hugging Face

Es una compañía y una plataforma comunitaria central para el procesamiento del lenguaje natural (PNL) y el aprendizaje automático. Proporciona una amplia gama de recursos, incluyendo modelos preentrenados (como los de la familia Llama), conjuntos de datos, bibliotecas (como Transformers), herramientas y una plataforma para colaborar en proyectos de IA.

`https://huggingface.co/`

C. LM Studio

Es una aplicación de escritorio diseñada para ejecutar modelos de lenguaje grandes (LLMs) localmente en la computadora. Permite a los usuarios descargar, gestionar y experimentar con varios modelos, incluyendo los de la familia Llama, sin necesidad de una conexión a internet constante o recursos de servidor en la nube.

`https://lmstudio.ai/`

D. Aspectos Administrativos

- La entrega del primer proyecto se mueve al martes 29 de abril.
- La clase presencial del martes 22 de abril se traslada a virtual.
- Se puede omitir el análisis del overfitting y underfitting puesto que scikit-learn no da estos datos, así como la implementación del validation set.

II. Funciones de Activación

Las funciones de activación permiten controlar la salida de las neuronas en una red neuronal. Estas funciones introducen no linealidades necesarias para que las redes puedan aprender representaciones complejas. A continuación se repasan las funciones más utilizadas en la práctica:

A. Función Sigmoide

- Rango de activación entre 0 y 1.
- Función creciente, positiva y acotada.
- Su derivada es máxima en el centro y disminuye en los extremos, lo que puede provocar problemas de *vanishing gradient* (gradiente desvanecido).
- En valores extremos, la pendiente se vuelve muy pequeña, lo que dificulta la actualización de los pesos.

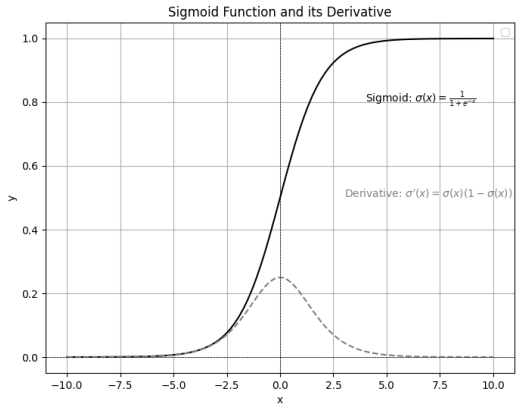

Fig. 1. Gráfica de la función sigmoide y su derivada

B. Función Tangente Hiperbólica (TanH)

- Rango entre -1 y 1.
- Similar a la sigmoide, pero centrada en el origen.
- También sufre del problema de gradiente desvanecido en los extremos.

- Se recomienda el uso de técnicas como batch normalization para mitigar este efecto.
- Común en modelos secuenciales como RNN y LSTM.

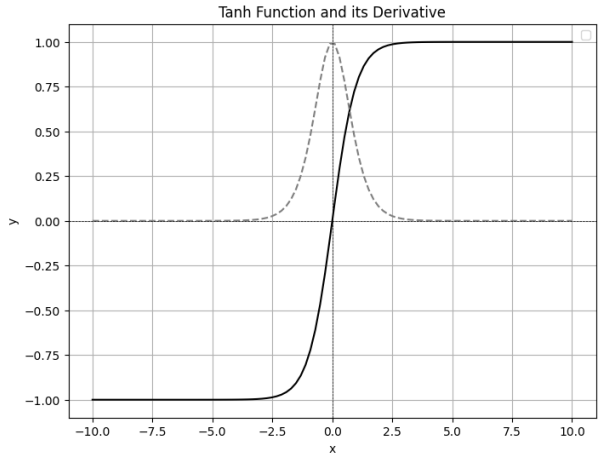

Fig. 2. Gráfica de la función tanH y su derivada

C. Función ReLU (Rectified Linear Unit)

- Define la salida como max(0, x).
- Función creciente, no acotada en valores positivos y cero en negativos.
- Computacionalmente eficiente y ampliamente utilizada en deep learning.
- Problema: neuronas "muertas" cuando la entrada es negativa, la derivada es cero, por lo que no se actualizan los pesos.

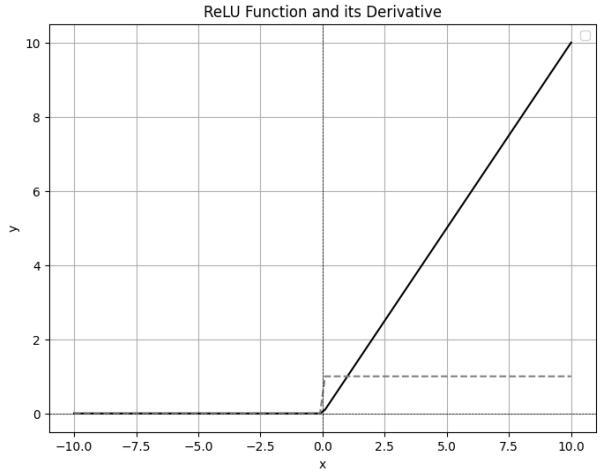

Fig. 3. Gráfica de la función ReLU y su derivada

D. Leaky ReLU

- Variante de ReLU que permite un pequeño gradiente cuando x < 0 (por ejemplo, 0.01x).
- Reduce el riesgo de neuronas muertas.
- El valor de la pendiente negativa (α) puede ser aprendido como un parámetro adicional: Parametric ReLU (PReLU).

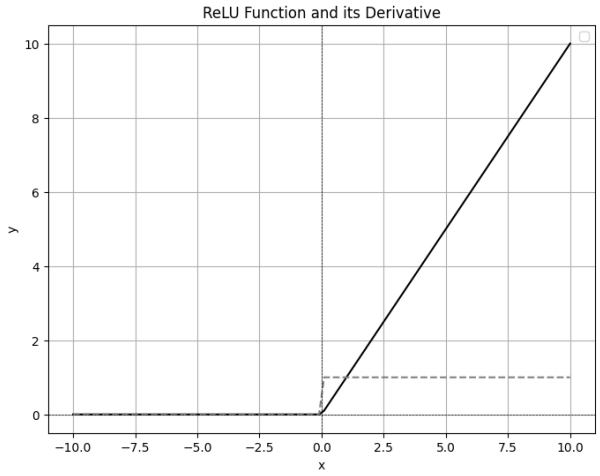

Fig. 4. Gráfica de la función Leaky ReLU y su derivada

E. Softmax

- Utilizada para problemas de clasificación multiclase.
- Convierte un vector de valores en una distribución de probabilidad.
- Se aplica generalmente en la capa de salida junto con la función de pérdida cross-entropy.
- Los valores de entrada a la función softmax se denominan logits.

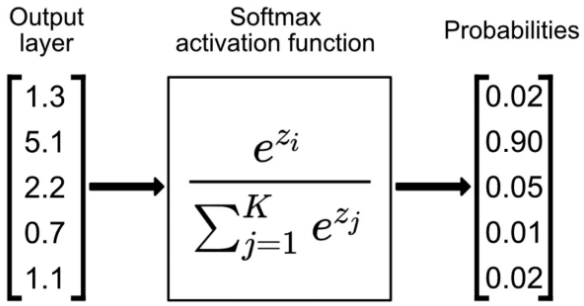

Fig. 5. Explicación de la función softmax

F. Consideraciones Prácticas

- ReLU es la opción por defecto para capas ocultas, pero no se recomienda en capas de salida.
- Evitar Sigmoide y TanH en redes profundas por problemas de *vanishing gradient*.
- Si ReLU no funciona se puede probar con sus variantes.
- TanH es usada en RNN o LSTM.
- En tareas de regresión, la capa de salida suele ser una neurona sin función de activación (lineal).

III. Función de Costo y Pérdida

A. Cross-Entropy Loss

$$L_i = -\log(P(Y = y_i, X = x_i\ )) \tag{1}$$

$$L_i = -\log(\frac{e^s_k}{\sum e^s_j}),\ \text{con } j \text{ desde } 1, .., C \tag{2}$$

- Mide la diferencia entre la distribución predicha por softmax y la real.
- Se penalizan más los errores con mayor diferencia entre predicción y valor real.
- Es una función estrictamente creciente, lo que permite una optimización más eficiente mediante gradiente.

IV. Backpropagation

El algoritmo de Backpropagation permite calcular la contribución de cada peso al error final de la red, actualizando los parámetros en sentido contrario a la propagación hacia adelante.

A. Procesos del Entrenamiento

1) Forward Propagation: Se calcula la salida de la red pasando los datos desde la entrada hacia las capas posteriores.
2) Backpropagation: Se propaga el error desde la capa de salida hacia las capas anteriores, calculando derivadas parciales respecto a los pesos y sesgos.

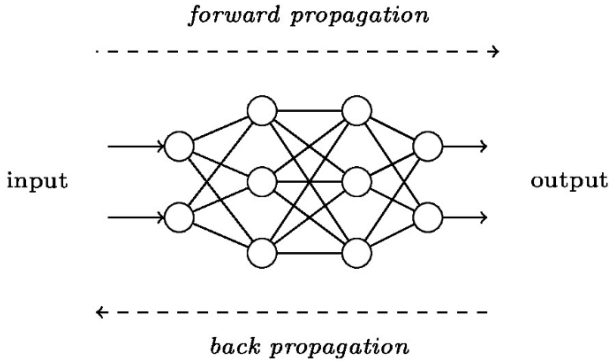

Fig. 6. Forward y Back Propagation

B. Optimización del grafo

Para este primer ejemplo tenemos una red donde cada capa contiene solo una neurona asumiendo que la función de activación es la Sigmoide, como se observa en la figura 7.

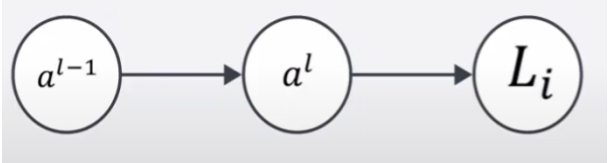

Fig. 7. Grafo de la red neuronal

- Denominamos a las capas antes de L_i, a^l hasta a^{l-n}
- Definimos el MSE $L_i = (a^l - y_i)^2$
- Dividimos la neurona en 2 capas:
  1) Entrada: $z^l = w^l a^{l-1} + b^l$ donde a^{l-1} corresponde a los inputs x
  2) Salida: $a^l = g(z^l)$ donde g es nuestra función de activación

Vamos a actualizar los parámetros del z^l, que son w^l y b^l. Para esto vamos a emplear la regla de la cadena usando la salida de la activación de la capa anterior. Profundizando a nivel de neurona tenemos la siguiente figura:

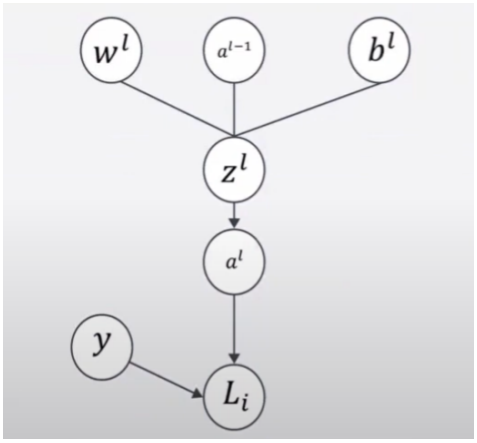

Fig. 8. Grafo de la capa a^l y L_i a detalle

Dado que necesitamos la derivada: $\frac{\delta L_i}{\delta w^l}$ debemos de pasar por cada uno de los nodos hasta llegar a los parámetros por regla de la cadena. Esto se ve de la siguiente forma:

$$\frac{\delta L_i}{\delta w^l} = \frac{\delta L_i}{\delta a^l}\frac{\delta a^l}{\delta z^l}\frac{\delta z^l}{\delta w^l} \tag{3}$$

$$\frac{\delta L_i}{\delta a^l} = 2(a^l - y) \tag{4}$$

$$\frac{\delta a^l}{\delta z^l} = g(z^l)(1 - g(z^l) \tag{5}$$

$$\frac{\delta z^l}{\delta w^l} = a^{l-1} \tag{6}$$

Mismo efecto se realiza para calcular b^l.

$$\frac{\delta z^l}{\delta b^l} = 1 \tag{7}$$

Hay cálculos que se repiten, por ende para lo obtención de parámetros estos son cacheados.

C. Vector Gradiente

Cabe destacar el proceso anterior nos calcula la última capa de activación respecto a la salida, teniendo en cuenta que esa capa solo tiene una neurona. Por ende, al momento de aumentar las dimensiones es necesario realizar ese calculo de las derivadas para todas las capas y a todas las neuronas que existan, a esto se le conoce como el vector gradiente.

Consiste en el calculo de todas las derivadas parciales respecto a la función de perdida.

$$\nabla L = \begin{bmatrix} \frac{\partial(L)}{\partial w^{(1)}} \\ \frac{\partial(L)}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial(L)}{\partial w^{(n)}} \\ \frac{\partial(L)}{\partial b^{(n)}} \end{bmatrix}$$

Fig. 9. Vector Gradiente

D. Múltiples Neuronas

Algunas consideraciones respecto a la nomenclatura a tomar en cuenta son las siguientes:

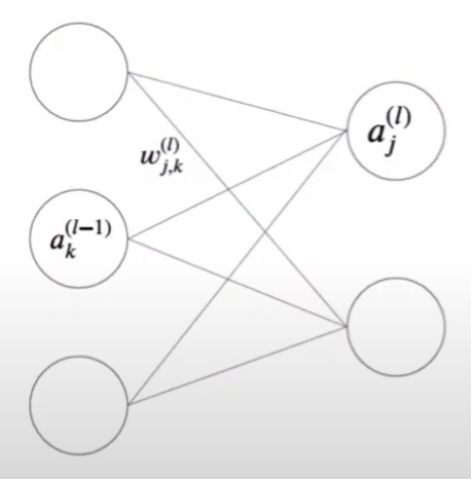

Fig. 10. Grafo con mayor dimensionalidad

- Superíndice: Indica la capa a la que pertenece. Ejemplo: $a^{(l)}$ pertenece a la capa l
- Subíndice: Indica el número de neurona dentro de una determinada capa. Ejemplo: $a_j^{(l)}$ pertenece a la capa l y la j-ésima neurona dentro de esa capa.
- Pesos: Para los pesos vamos a usar dos subíndices, el primero para la neurona destino y el segundo para la neurona de donde proviene. Ejemplo: $w_{j,k}^{(l)}$ se refiere a que es el peso que conecta desde la neurona $a_k^{(l-1)}$ hasta la neurona $a_j^{(l)}$.

A continuación veremos dos casos prácticos de cálculos con múltiples neuronas.

1) De k neuronas a 1: En el caso de tener k neurona en la capa $l-1$ asociadas a una neurona en la capa l tenemos que modificar las funciones de preactivación y activación.

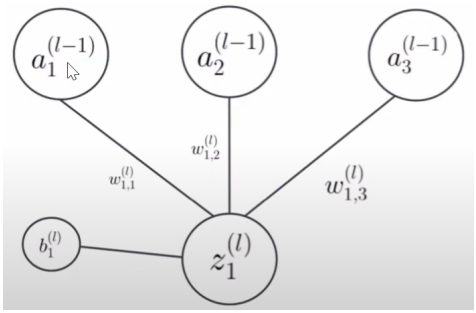

Fig. 11. Grafo donde 3 neurona conectan con 1

- Preactivación:

$$z_j^{(l)} = b_j^{(l)} + \sum_{k=1}^{n_{l-1}} w_{j,k}^{(l)} a_k^{(l-1)} \tag{8}$$

  Donde n_{l-1} corresponde al número de neurona de la capa $l-1$. Al final es el mismo producto punto que calculamos siempre, solo hay que tener cuidado de donde vienen los pesos.
- Activación:

$$a_j^{(l)} = g(z_j^{(l)}) \tag{9}$$

2) De k neuronas a j: Para el caso en el que la capa l tenga j neuronas, se debe aplicar el procesamiento anterior (preactivación, activación) para cada uno de los j de esa capa.

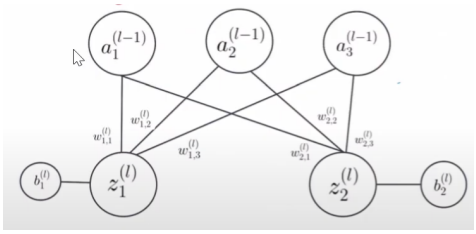

Fig. 12. Grafo donde 3 neurona conectan con 2

E. Cálculo de función de perdida

Para esta sección se calcula el *loss* global mediante la sumatoria de la salida de la capa de activación j menos el y_j recorriendo así cada una de las neuronas de la capa l.

$$L_i = \sum_{j=1}^{n_l} (a_j^{(l)} - y_j)^2 \tag{10}$$

En la siguiente tabla muestra un ejemplo de la salida de una capa de activación siendo evaluada con el *loss function*.

TABLE I
EJEMPLO

Y_pred	Target	Loss
0.8	1	0.039
0.2	0	0.040
		0.079

F. Cambios a la Regla de la Cadena

Dado que las funciones L_i, $z_j^{(l)}$ y $a_j^{(l)}$ han cambiado, es necesario plantear nuevas derivadas para poder actualizar los parámetros k de cada neurona j.

$$\frac{\delta L_i}{\delta w_{j,k}^{(l)}} = \frac{\delta z_j^{(l)}}{\delta w_{j,k}^{(l)}} \frac{\delta a_j^{(l)}}{\delta z_j^{(l)}} \frac{\delta L_i}{\delta a_j^{(l)}} \tag{11}$$

Podemos observar en la ecuación que para actualizar cada peso $w_{j,k}^{(l)}$ especifico es necesario calcular esas derivadas parciales. Continuando con el concepto de caché, la unica derivada que va a cambiar al actualizar un peso diferente es $\delta w_{j,k}^{(l)}$, el resto se mantienen igual para toda la capa. A continuación resolveremos esas derivadas:

$$\frac{\delta L_i}{\delta a_j^l} = ((a_1^l - y_1)^2 + (a_2^l - y_2)^2 + ... + (a_n^l - y_n)^2)$$

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

Como se deriva con respecto a un j especifico, los demás términos se consideran constantes."

$$\frac{\delta a_j^{(l)}}{\delta z_j^{(l)}} = g(z_j^{(l)})(1 - g(z_j^{(l)})) \tag{12}$$

$$\frac{\delta z_j^{(l)}}{\delta w_{j,k}^{(l)}} = a_k^{(l-1)} \tag{13}$$

Con esto ya habríamos logrado actualizar los pesos de la capa l, los cuales no cambian las derivadas, pero si manejan mas indices. Ahora que pasa cuando yo necesito calcular una capa anterior como la capa $l-1$.

G. Capa a^{l-1}

Sin entrar mucho en detalles esta es la formula.

$$\frac{\delta L_i}{\delta a_k^{(l-1)}} = \sum_{j=1}^{n_l} \frac{\delta z_j^{(l)}}{\delta a_k^{(l-1)}} \frac{\delta a_j^{(l)}}{\delta z_j^{(l)}} \frac{\delta L_i}{\delta a_j^{(l)}} \tag{14}$$

A grandes rasgos podemos observar lo complicado que se vuelve el algoritmo. De acuerdo al tamaño de la siguiente capa el problema se vuelve mucho más complejo.