UACM
Universidad Autónoma
de la Ciudad de México
NADA HUMANO ME ES AJENO

Redes Neuronales

Sabino Miranda

Funciones de activación

Funciones de activación

- Las funciones de activación son funciones matemáticas aplicadas a la salida de cada neurona en una red neuronal y tienen un papel esencial en el aprendizaje y el modelado de relaciones complejas en los datos.
- Su propósito principal es **introducir no linealidad en el modelo**, permitiendo que la red neuronal aprenda patrones no lineales.
- Sin una función de activación, una red neuronal profunda sería esencialmente una combinación lineal de sus entradas y pesos, lo cual limitaría su capacidad para resolver problemas complejos.
- La función de activación permite que la red aprenda y modele relaciones complejas en los datos.

Características clave de las Funciones de activación

- **No linealidad**: Permite que la red neuronal aprenda y represente relaciones no lineales en los datos.
- **Continuidad y derivabilidad**: Para permitir el aprendizaje a través del algoritmo de retropropagación (backpropagation), la función de activación debe ser continua y diferenciable (en la mayoría de los casos).
- **Control de la saturación**: Algunas funciones de activación limitan el rango de salida (por ejemplo, sigmoide y tanh), ayudando a evitar activaciones excesivamente grandes.

Funciones de activación y su gradiente (1)

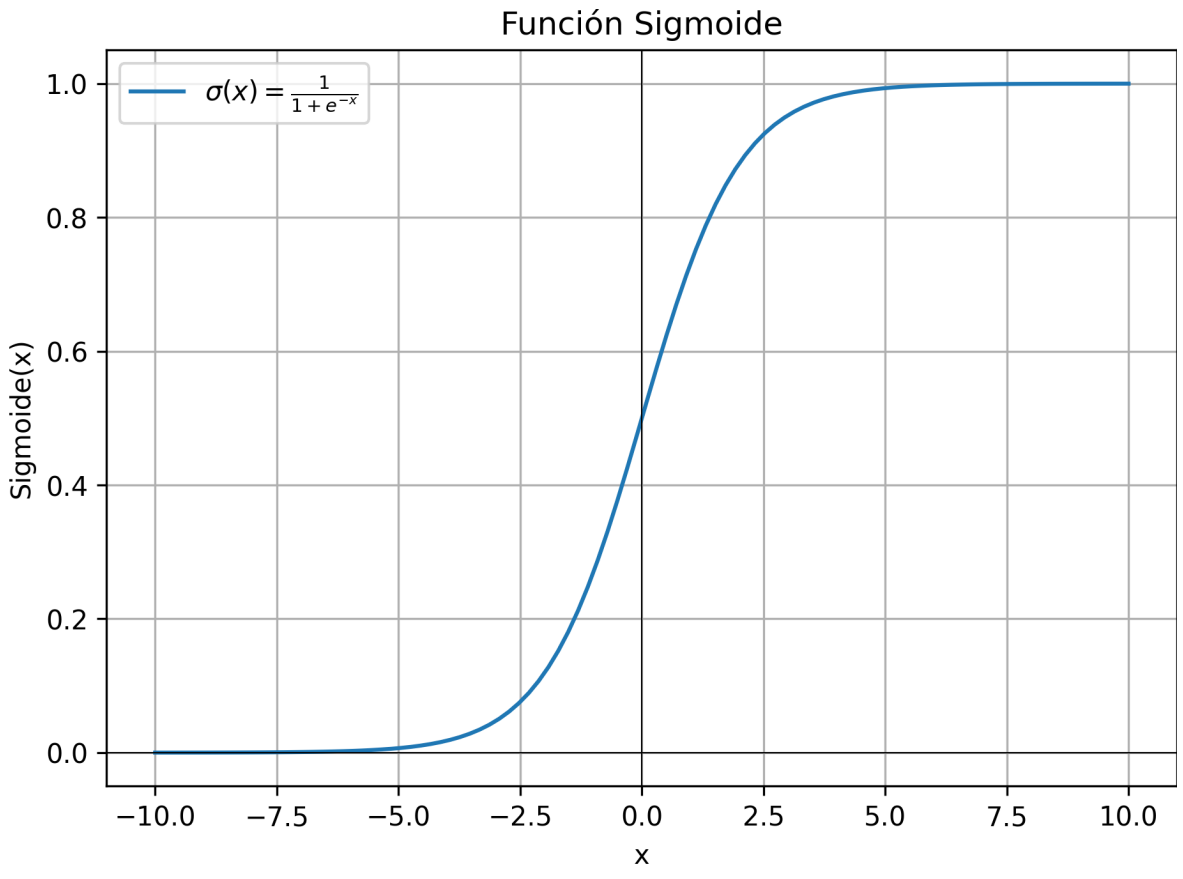
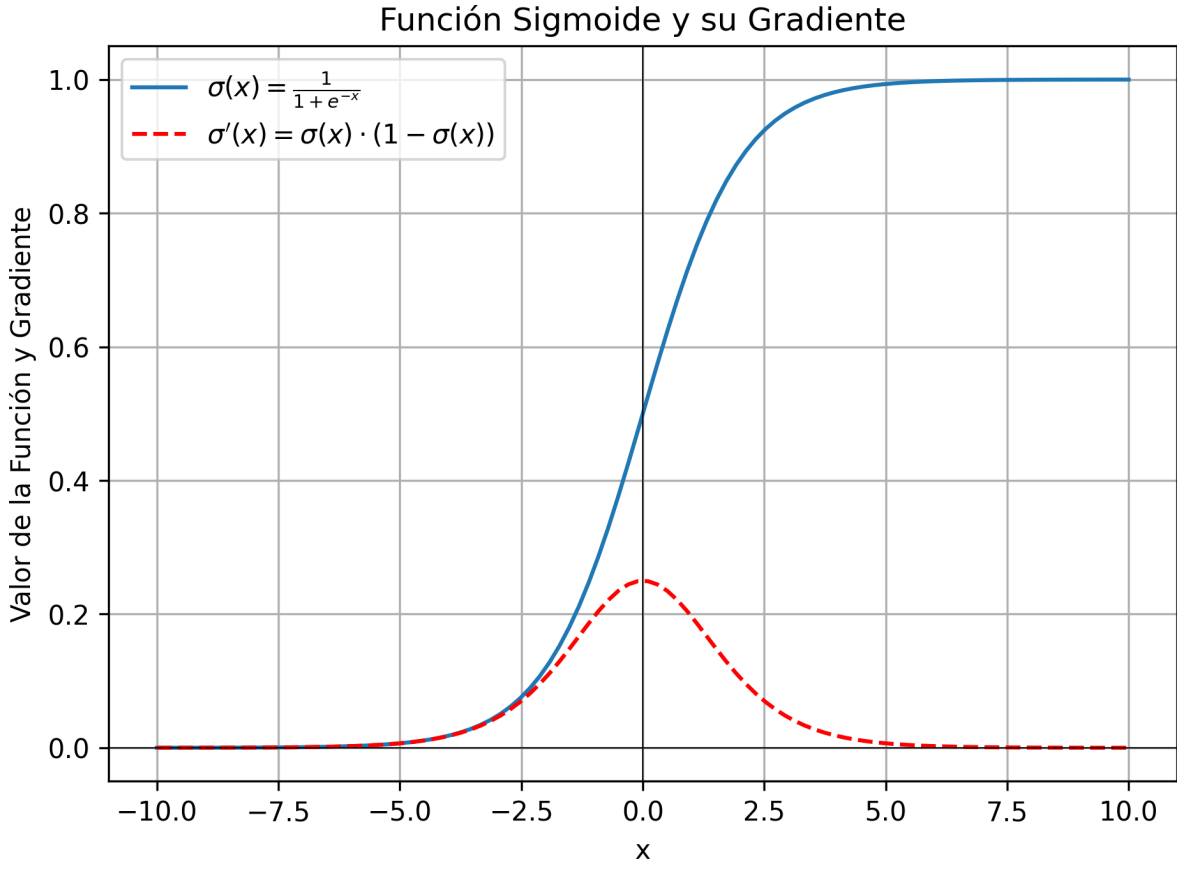


Figura: Función de activación Sigmoide: $\sigma(x) = \frac{1}{1+e^{-x}}$

Funciones de activación y su gradiente (2)



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

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

Funciones de activación y su gradiente (3)

- Por ejemplo en PyTorch:

```
import torch
from torch import nn
sigmoide = nn.Sigmoid()
x = torch.tensor( [-5.0, 0, 5.0], requires_grad=True)
y = sigmoide(x)
print(y)
---
tensor([0.0067, 0.5000, 0.9933], grad_fn=<SigmoidBackward0>)
```

Funciones de activación y su gradiente (4)

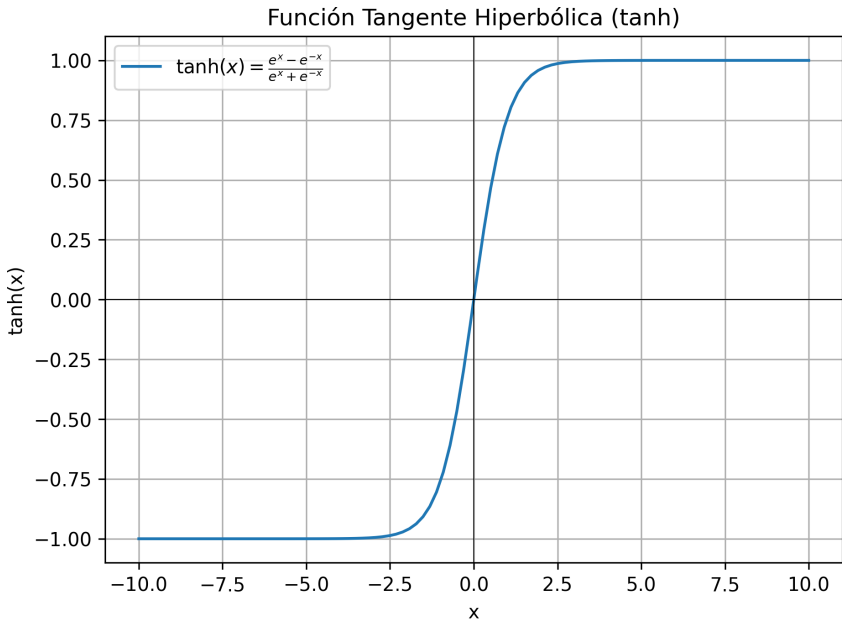
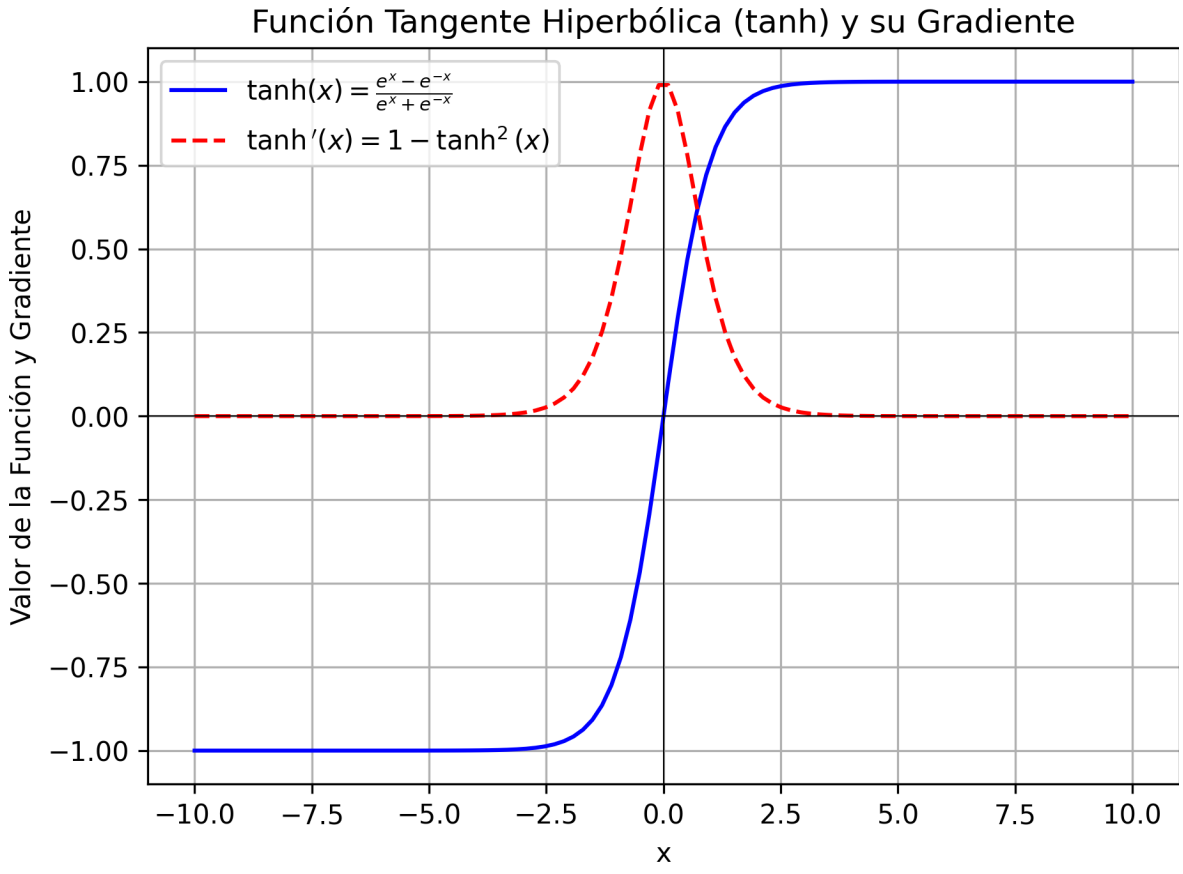


Figura: Función de activación tangente hiperbólica:

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

Funciones de activación y su gradiente (5)



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

$$\tanh'(x) = 1 - \tanh^{2}(x)$$

Funciones de activación y su gradiente (6)

- Por ejemplo en PyTorch:

```
import torch
from torch import nn
tanh = nn.Tanh()
x = torch.tensor( [-5.0, 0, 5.0], requires_grad=True)
y = tanh(x)
print(y)
---
tensor([-0.9999,  0.0000,  0.9999], grad_fn=<TanhBackward0>)
```

Funciones de activación y su gradiente (7)

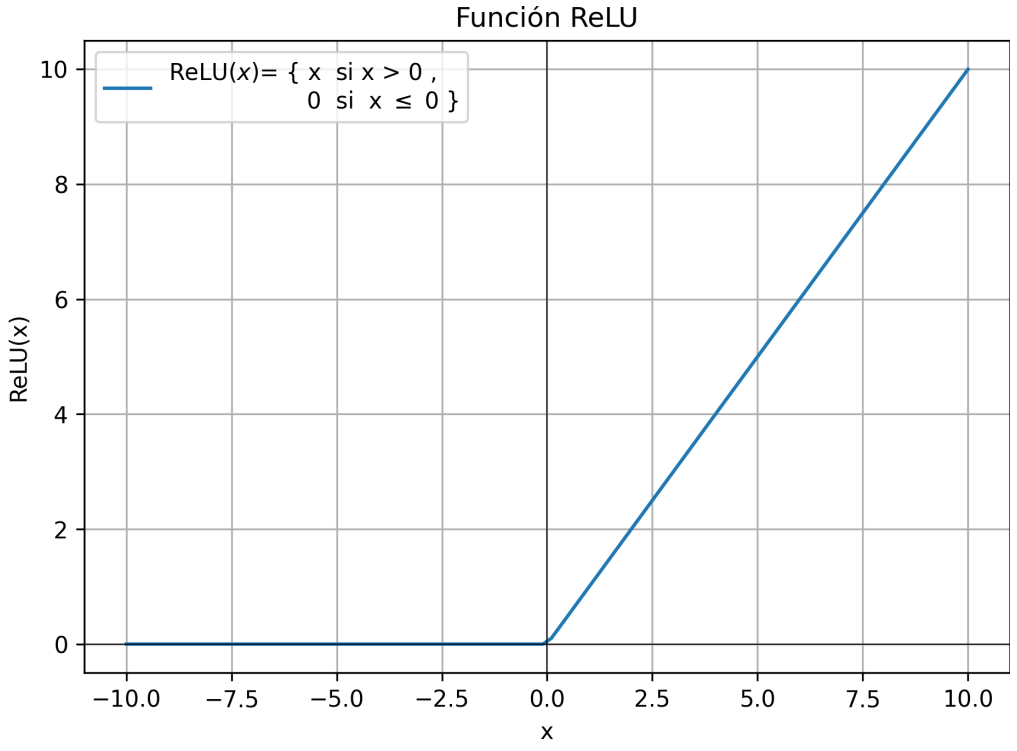
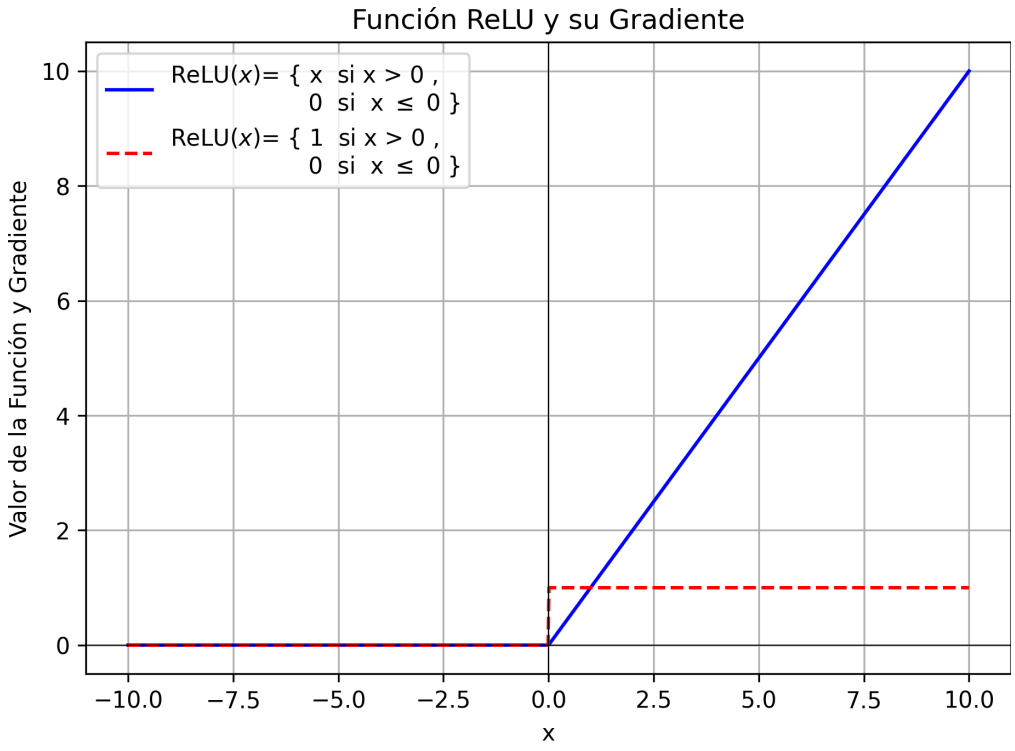


Figura: Función de activación Rectified Linear Unit (ReLU):

$$\text{ReLU}(x) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases}$$

Funciones de activación y su gradiente (8)



Funciones de activación y su gradiente (9)

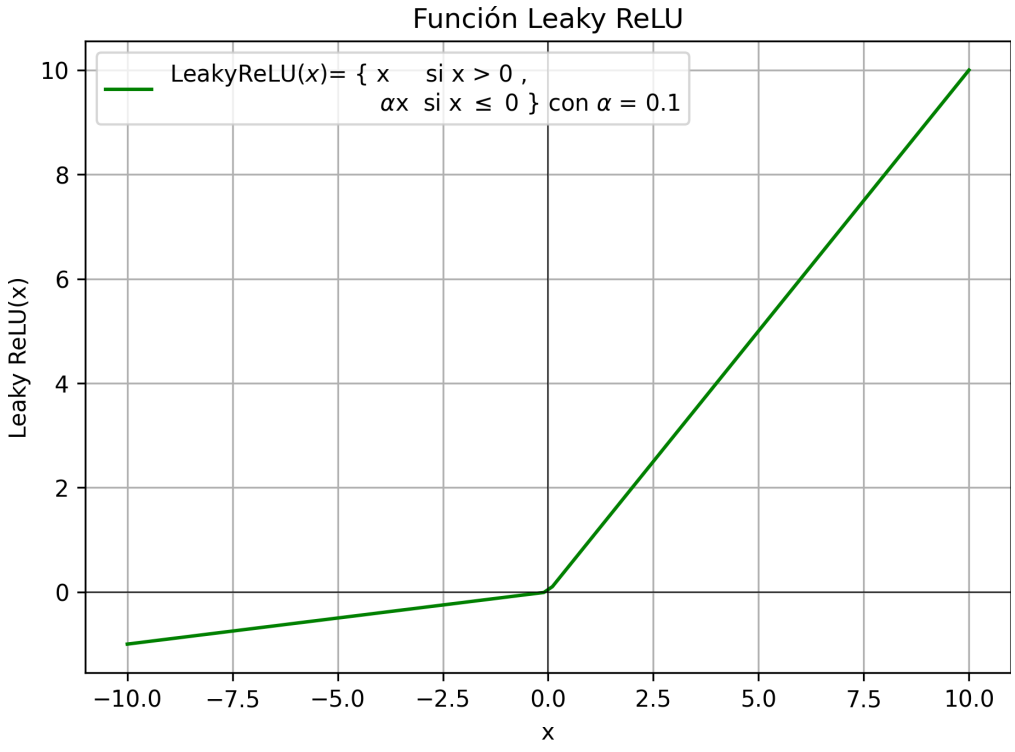
$$\text{ReLU}(x) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases}$$

$$\text{ReLU}'(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases}$$

- Por ejemplo en PyTorch:

```
import torch
from torch import nn
relu = nn.ReLU()
x = torch.tensor( [-5.0, 0, 5.0], requires_grad=True)
y = relu(x)
print(y)
---
tensor([0., 0., 5.], grad_fn=<ReluBackward0>)
```

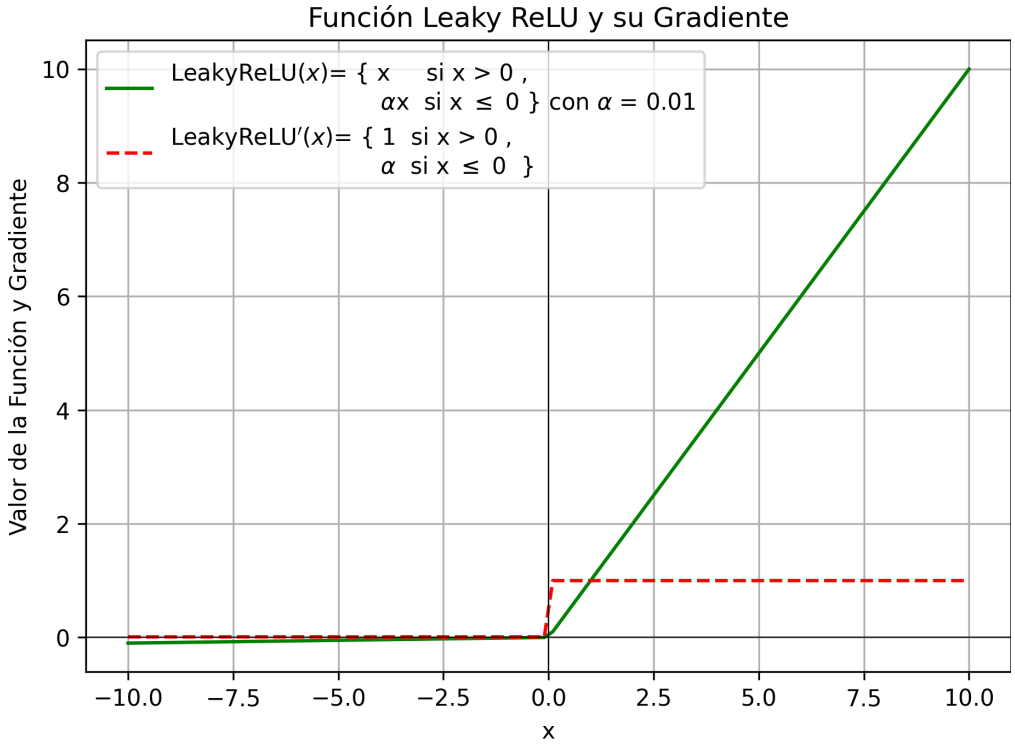
Funciones de activación y su gradiente (10)



$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha x & \text{si } x \leq 0 \end{cases}$$

α es fija.

Funciones de activación y su gradiente (11)



Funciones de activación y su gradiente (12)

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha x & \text{si } x \leq 0 \end{cases}$$

$$\text{LeakyReLU}'(x) = \begin{cases} 1 & \text{si } x > 0 \\ \alpha & \text{si } x \leq 0 \end{cases}$$

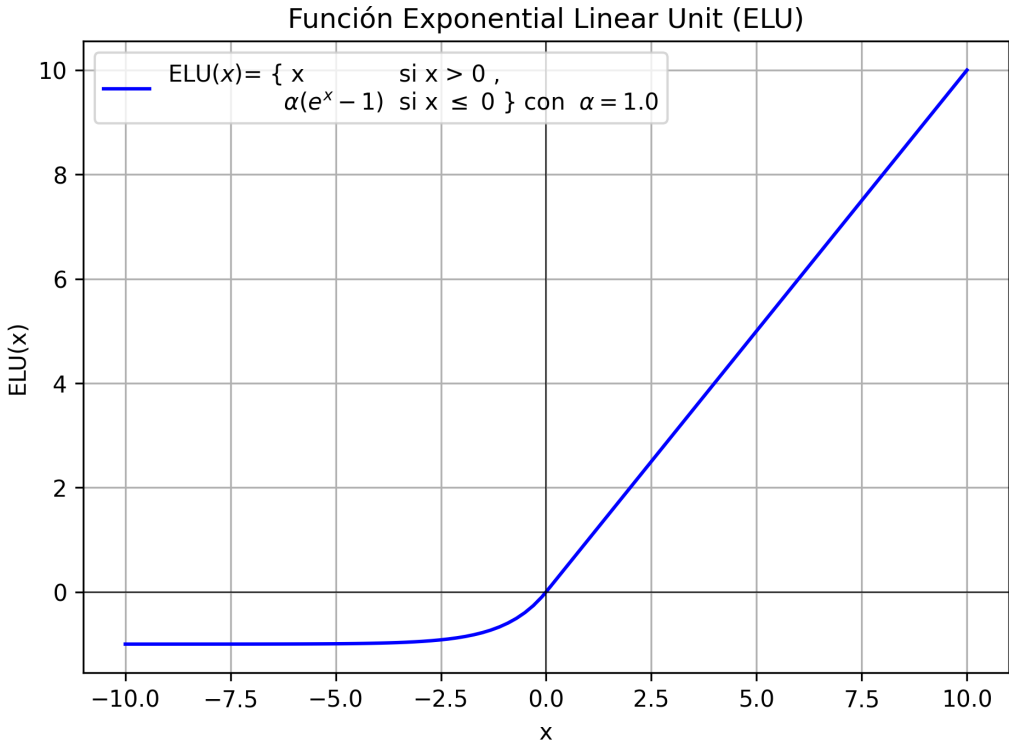
α es fija.

Funciones de activación y su gradiente (13)

- Por ejemplo en PyTorch:

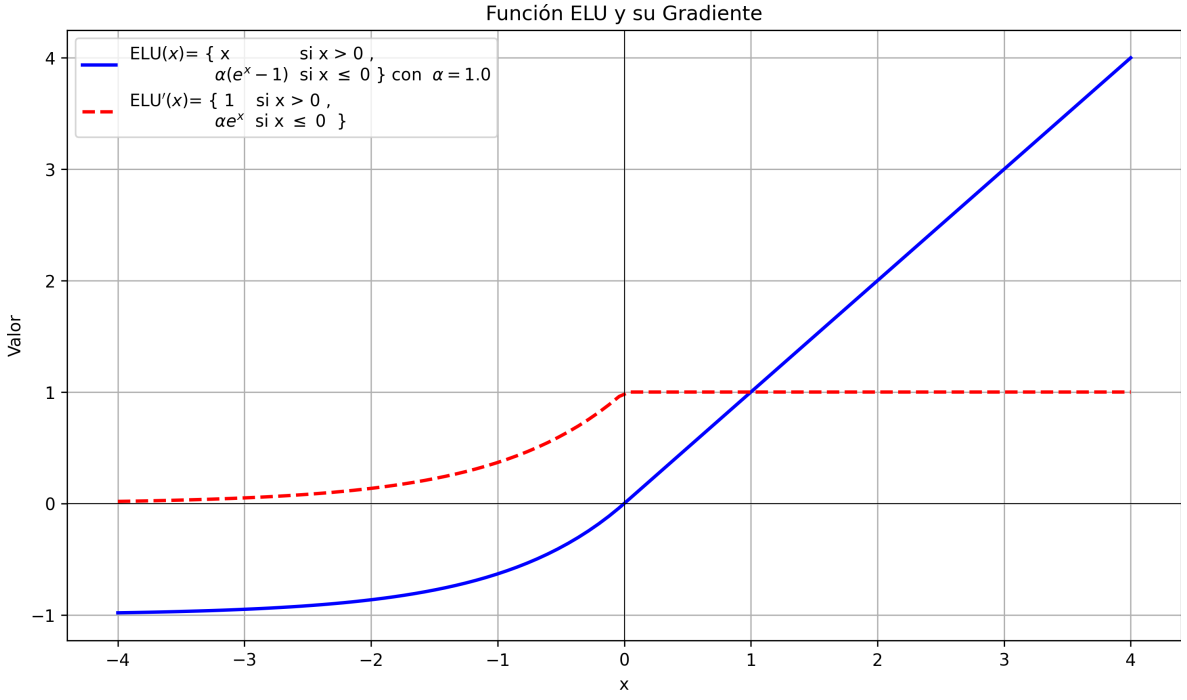
```
import torch
from torch import nn
# negative_slope = alfa
laekyrelu = nn.LeakyReLU(negative_slope=0.3)
x = torch.tensor( [-5.0, 0, 5.0], requires_grad=True)
y = laekyrelu(x)
print(y)
---
tensor([-1.5000,  0.0000,  5.0000],
↪  grad_fn=<LeakyReluBackward0>)
```

Funciones de activación y su gradiente (14)



$$\text{ELU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha\,(e^x - 1) & \text{si } x \leq 0 \end{cases}$$

Funciones de activación y su gradiente (15)



$$\mathrm{ELU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha\,(e^{x} - 1) & \text{si } x \leq 0 \end{cases}$$

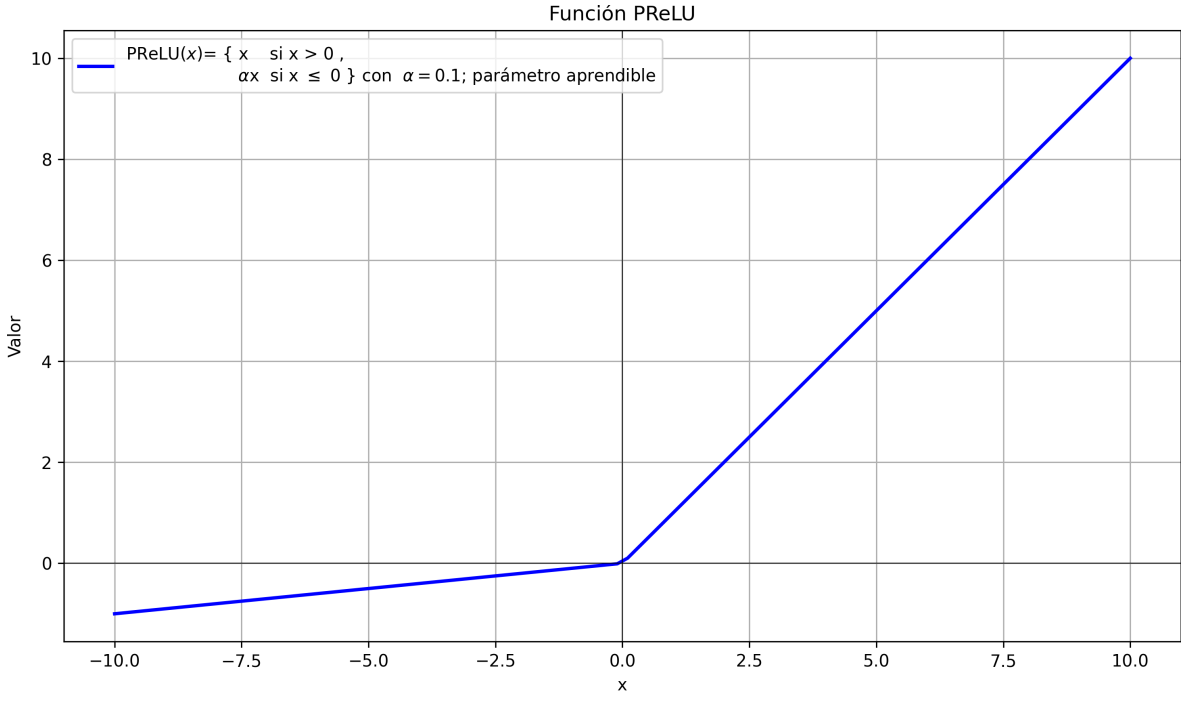
$$\mathrm{ELU}'(x) = \begin{cases} 1 & \text{si } x > 0 \\ \alpha e^{x} & \text{si } x \leq 0 \end{cases}$$

Funciones de activación y su gradiente (16)

- Por ejemplo en PyTorch:

```
import torch
from torch import nn
elu = nn.ELU(alpha=1)
x = torch.tensor( [-5.0, 0, 5.0], requires_grad=True)
y = elu(x)
print(y)
---
tensor([-0.9933,  0.0000,  5.0000], grad_fn=<EluBackward0>)
```

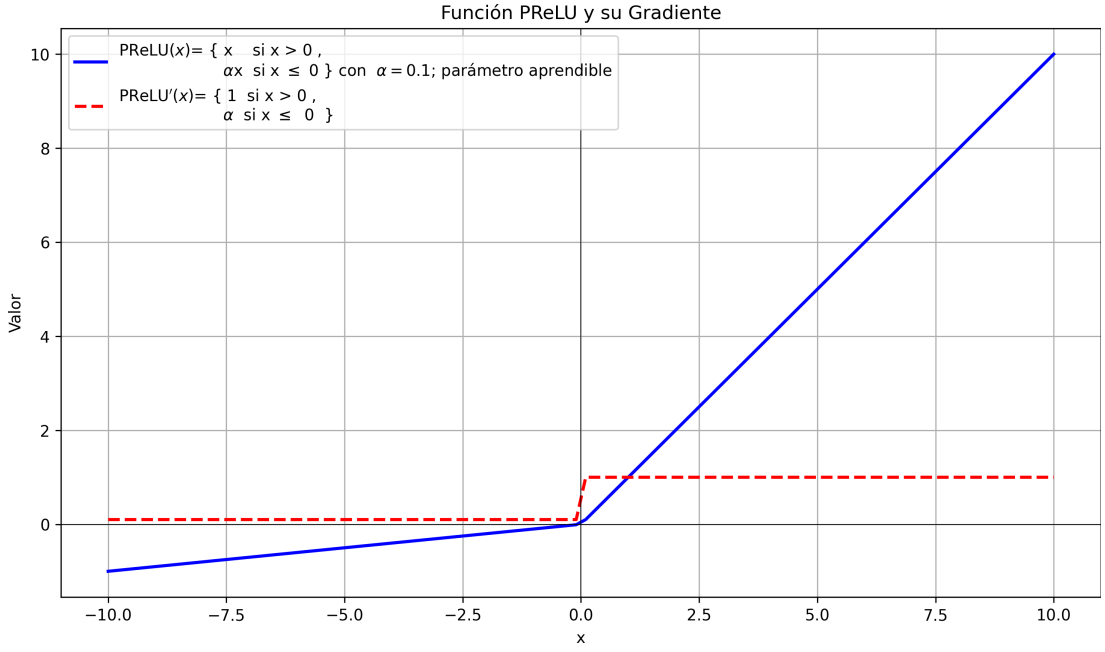
Funciones de activación y su gradiente (17)



$$\text{PReLU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha x & \text{si } x \leq 0 \end{cases}$$

α es un parámetro que se optimiza (se aprende).

Funciones de activación y su gradiente (18)



$$\text{PReLU}(x) = \begin{cases} x & \text{si } x > 0 \\ \alpha x & \text{si } x \leq 0 \end{cases}$$

$$\text{PReLU}'(x) = \begin{cases} 1 & \text{si } x > 0 \\ \alpha & \text{si } x \leq 0 \end{cases}$$

α es un parámetro que se optimiza (se aprende).

- Por ejemplo en PyTorch:

Funciones de activación y su gradiente (19)

```
import torch
from torch import nn
# El parámetro alfa se aprende durante el entrenamiento
# alfa, con parámetro inicial: init=0.25 por default
prelu = nn.PReLU() # nn.PReLU(init=0.25)

x = torch.tensor( [-5.0, 0, 5.0], requires_grad=True)
y = prelu(x)
print(y)

---
tensor([-1.2500,  0.0000,  5.0000],
↪  grad_fn=<PreluKernelBackward0>)
```

Consideraciones: Funciones de activación (1)

- Funciones de activación como ReLU, PReLU, Leaky ReLU, etc. o tipo sigmoide son preferidas porque introducen no linealidades asimétricas (parte positiva y negativa), lo cual permite que las redes aprendan funciones complejas.

Consideraciones: Funciones de activación (2)

- Función sigmoide
  - Ventajas
    - Útil en clasificación binaria ya que produce una salida entre 0 y 1, interpretada como probabilidad.
    - Se recomienda en capa de salida para clasificación binaria.
  - Desventajas
    - Sufre de desvanecimiento del gradiente para redes profundas (el gradiente se hace muy pequeño) en capas ocultas.
    - No está centrada en cero, lo que puede hacer que el entrenamiento sea más lento.
    - Cálculo computacional costoso.

Consideraciones: Funciones de activación (3)

- Función Tanh
  - Ventajas
    - Centrada en cero, lo que puede mejorar la convergencia de la red.
    - Suaviza el gradiente
    - Se puede usar para clasificación binaria
  - Desventajas
    - Sufre de desvanecimiento del gradiente para redes profundas (el gradiente se hace muy pequeño) en capas ocultas.
    - Cálculo computacional costoso.

Consideraciones: Funciones de activación (4)

- Función ReLU
  - Ventajas
    - Simple y eficiente computacionalmente.
    - Previene el desvanecimiento del gradiente.
    - Se recomienda en capas ocultas para redes profundas: MLP, Redes convolucionales y problemas de clasificación que no requieran la activación negativa.
  - Desventajas
    - No tiene salida negativas
    - Puede producir neuronas muertas (cuando $x \leq 0$)

Consideraciones: Funciones de activación (5)

- Función Leaky ReLU
  - Ventajas
    - No produce neuronas muertas
    - Se tiene control de la pendiente para la parte negativa α, es fijo.
    - Se recomienda en capas ocultas para redes profundas con activaciones negativas.
  - Desventajas
    - Puede ser menos eficiente que ReLU.
    - Se requiere ajustar el parámetro α.

Consideraciones: Funciones de activación (6)

- Función ELU
  - Ventajas
    - Mejora el aprendizaje con valores negativos.
    - Suaviza la activación negativa.
    - Se recomienda en redes profundas, cuando se necesita una activación negativa suavizada.
  - Desventajas
    - Cálculo más costoso.
    - Sensibilidad al parámetro α.

Consideraciones: Funciones de activación (7)

- Función PReLU
  - Ventajas
    - Similar a Leaky ReLU, pero α se ajusta por el modelo.
    - Evita neuronas muertas.
    - Se recomienda en redes profundas, cuando se requiere un control más fino de la activación negativa.
  - Desventajas
    - Más parámetros que se deben de ajustar.
    - Requiere más datos y tiempo para el entrenamiento.

Función de activación Softmax (1)

- Se utiliza comúnmente como función de activación para la última capa de la red neuronal en el caso de clasificación de múltiples clases.
- La función `softmax` para un vector de entradas $\mathbf{z} = [z_1, z_2, \ldots, z_n]$ se define como:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \quad \text{para} \quad i = 1, 2, \ldots, n$$

Donde:

- z_i es el valor de la i-ésima entrada en el vector $\mathbf{z}$,
- n es el número total de clases o entradas,

Función de activación Softmax (2)

- La salida $\text{softmax}(\mathbf{z})_i$ es la probabilidad asociada a la clase i,
- La función softmax convierte las salidas en un rango de probabilidades que suman 1:

$$\sum_{i=1}^{n} \text{softmax}(\mathbf{z})_i = 1.$$

Función de activación Softmax (3)

- Supongamos que el vector de entradas es $\mathbf{z} = [2.0, 1.0, 0.1]$. La salida de la función softmax es:

$$\text{softmax}(\mathbf{z}) = \left[\frac{e^{2.0}}{e^{2.0} + e^{1.0} + e^{0.1}}, \frac{e^{1.0}}{e^{2.0} + e^{1.0} + e^{0.1}}, \frac{e^{0.1}}{e^{2.0} + e^{1.0} + e^{0.1}} \right]$$

$$\text{softmax}(\mathbf{z}) = [0.6590, 0.2424, 0.0986]$$

Función de activación Softmax (4)

- Por ejemplo en PyTorch:

```
import torch
from torch import nn
softmax = nn.Softmax(dim=1)
x = torch.tensor( [[2.0, 1.0, 4],
                   [5.0, 3.0, 5.0]],
                   ↪  requires_grad=True)
y = softmax(x)
print(y)
---
tensor([[0.1142, 0.0420, 0.8438],
        [0.4683, 0.0634, 0.4683]],
        ↪  grad_fn=<SoftmaxBackward0>)
```

Referencias (1)

1. Deep Learning. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. MIT Press, 2016. http://www.deeplearningbook.org
2. Dive into Deep Learning. Aston Zhang, Zachary C. Lipton, Mu li, and Alexander J. Smola. Cambridge University Press, 2023. https://d2l.ai
3. Neural Networks and Deep Learning A Textbook (2nd Edition). Charu C. Aggarwal. Springer, 2023. https://doi.org/10.1007/978-3-031-29642-0
4. Deep Learning: Foundations and Concepts. Christopher M. Bishop and Hugh Bishop. Springer, 2024. https://doi.org/10.1007/978-3-031-45468-4

Referencias (2)

5. PyTorch documentation.
   `https://pytorch.org`
6. Numpy documentation.
   `https://numpy.org`
7. Python documentation.
   `https://www.python.org`