Informe Profundizado: Funciones de Activación en Redes Neuronales

1. Introducción

**Las funciones de activación son componentes esenciales en las redes neuronales porque introducen no-linealidad, permitiendo que la red aprenda patrones complejos en los datos. Sin ellas, una red neuronal profunda sería equivalente a una sola capa lineal, incapaz de resolver problemas como:**

**Dato: analogías claras (como la "cobaya" que necesita ser cubierta) para ilustrar cómo las funciones de activación permiten a las redes neuronales modelar relaciones no lineales complejas.**

Reconocimiento de imágenes

Procesamiento de lenguaje natural

Predicción de series temporales

**En este informe, profundizaremos en:**

**✅ Tipos de funciones de activación (clásicas y modernas)**

**✅ Cómo afectan al entrenamiento (backpropagation, gradientes)**

**✅ Problemas comunes (neuronas muertas, vanishing gradient)**

**✅ Recomendaciones prácticas (cuándo usar cada una)**

1. **Propósito de las funciones de activación**: Agregar no-linealidad a las redes neuronales, permitiéndoles aproximar funciones complejas del mundo real.
2. **Problema sin funciones de activación**: Una red sin ellas solo puede representar transformaciones lineales, sin importar cuántas capas tenga.
3. **Backpropagation**: Las funciones deben ser diferenciables para permitir el cálculo de gradientes durante el entrenamiento.

2. Funciones de Activación Clásicas

A. Función Escalón (Step Function)

**Fórmula:**

f(x)={1 si x≥0, 0 si x<0}

Problema: No es diferenciable → no útil para backpropagation.

* **Concepto**: Devuelve 0 para valores negativos y 1 para positivos.
* **Problema**: No es diferenciable (derivada = 0 en casi todos lados).
* **No se usa en redes modernas**.

B. Sigmoide (Logística)

Fórmula: σ(x) = 1 / (1 + e^-x)

* **Rango**: (0, 1)
* **Problema**: Desvanecimiento de gradiente en valores extremos.
* **Uso actual**: Principalmente en capa de salida para clasificación binaria.

**Problema:**

- Vanishing gradient (derivada ≈ 0 para |x| grandes).

- Salidas no centradas en 0 → aprendizaje lento.

C. Tangente Hiperbólica (Tanh)

Fórmula: tanh(x) = (e^x − e^−x) / (e^x + e^−x)

Rango: (-1, 1)

Ventaja: Mejor que sigmoide porque centra las salidas en 0.

Problema: También sufre de vanishing gradient.

D. ReLU (Rectified Linear Unit)

Fórmula: ReLU(x) = max(0, x)

**Ventajas:**

- Computacionalmente eficiente.

- Evita vanishing gradient para x > 0.

Problema: Neuronas muertas (si x ≤ 0, la neurona "muere").

3. Funciones de Activación Modernas

A. Leaky ReLU

Fórmula: LeakyReLU(x)={x si x>0, αx si x≤0} (α ≈ 0.01)

Ventaja: Evita neuronas muertas.

Problema: α fijo puede no ser óptimo.

B. PReLU (Parametric ReLU)

Similar a Leaky ReLU, pero α se aprende durante el entrenamiento.

C. ELU (Exponential Linear Unit)

Fórmula: ELU(x)={x si x>0, α(e^x−1) si x≤0}

Ventaja: Suaviza el gradiente para x≤0.

**4. Funciones Modernas**

D. Swish (Google Brain, 2017)

Fórmula: Swish(x)=x⋅σ(βx), donde σ es la sigmoide

Ventaja: Mejor rendimiento que ReLU en redes profundas.

Costo: Más cara computacionalmente.

E. Mish (2020)

Fórmula: Mish(x)=x⋅tanh(ln(1+e^x))

Ventaja: Excelente en visión por computadora.

Usada en YOLOv4, pero reemplazada en YOLOv5 por ReLU.

F. GELU (Gaussian Error Linear Unit)

Fórmula: GELU(x)=x⋅Φ(x), donde Φ(x) es la función de distribución Gaussiana

Usada en Transformers (BERT, GPT).

4. Backpropagation y Neuronas Muertas

A. Backpropagation y Derivadas

Las funciones de activación deben ser diferenciables para calcular gradientes.

**Ejemplo con ReLU:**

d/dx ReLU(x) = {1 si x>0, 0 si x≤0}

Si x≤0, el gradiente es 0 → neurona muerta.

B. ¿Por qué mueren las neuronas con ReLU?

- Inicialización incorrecta: Pesos muy negativos hacen que z=w⋅x+b≤0.

- Learning rate alto: Los pesos "saltan" a valores que siempre dan z≤0.

- Ruido en los datos: Algunas neuronas se saturan en negativo.

C. Soluciones para Neuronas Muertas

**✅ Usar Leaky ReLU / PReLU / Swish (evitan gradiente cero).**

**✅ Batch Normalization (estabiliza las activaciones).**

**✅ Ajustar learning rate (evitar saltos bruscos).**

**✅ Inicialización He/Xavier (ajusta la escala inicial de los pesos).**

Ejemplo Matemático Simplificado:

Para una neurona con:

* Entrada: x
* Peso: w
* Bias: b
* Función de activación: σ (ej. ReLU)
* Salida: y = σ(wx + b)

Durante backpropagation:

1. Calculamos ∂L/∂y (gradiente de la pérdida respecto a la salida)
2. Calculamos ∂y/∂z donde z = wx + b (esto es la derivada de la función de activación)
3. Propagamos el gradiente: ∂L/∂w = ∂L/∂y \* ∂y/∂z \* ∂z/∂w = ∂L/∂y \* σ'(z) \* x

**La derivada de la función de activación (σ') es crucial**:

* En ReLU: σ'(z) = 1 si z > 0, 0 si z ≤ 0
* En Leaky ReLU: σ'(z) = 1 si z > 0, α (pequeño) si z ≤ 0

Problema del Vanishing Gradient:

Ocurre cuando:

* Las derivadas son muy pequeñas (ej. en sigmoid para valores extremos)
* Los gradientes se multiplican a través de muchas capas
* El producto final es extremadamente pequeño
* Los pesos en las primeras capas casi no se actualizan

Neuronas Muertas - Explicación Ampliada

¿Qué son las neuronas muertas?

Son neuronas que siempre producen salida 0 (en ReLU) o un valor constante, dejando de contribuir al aprendizaje.

Causas principales:

1. **Inicialización inadecuada de pesos**:
   * Pesos iniciales demasiado grandes/pequeños
   * Puede causar que muchas neuronas empiecen en estado "muerto"
2. **Tasa de aprendizaje (learning rate) muy alta**:
   * Las actualizaciones son tan grandes que los pesos "saltan" a valores que siempre producen z ≤ 0
   * Ejemplo: Si w se vuelve muy negativo, wx + b ≤ 0 para todas las entradas x
3. **Bias negativo grande**:
   * El término b en wx + b domina y hace que z sea siempre ≤ 0
4. **Distribución de datos**:
   * Si ciertas características son siempre cero o muy pequeñas

Consecuencias:

* Reducción de la capacidad de la red
* Pérdida de información
* Subutilización de recursos computacionales
* Estancamiento en el aprendizaje

Código ejemplo , neurona muerta

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | **import** **numpy** **as** **np**  *# Neurona con ReLU*  **def** relu\_neuron(x, w, b):  z = np.dot(x, w) + b  **return** np.maximum(0, z)  *# Datos de ejemplo*  x = np.array([0.5, -1.2, 0.8])  w = np.array([-2.0, -1.5, -3.0]) *# Pesos muy negativos*  b = -0.5  *# Forward pass*  output = relu\_neuron(x, w, b)  **print**(f"Salida: {output}") *# Será 0 (neurona muerta)*  *# Backward pass*  dL\_dy = 1.0 *# Supongamos gradiente inicial de 1*  dy\_dz = 1 **if** np.dot(x, w) + b > 0 **else** 0 *# Derivada ReLU*  dL\_dw = dL\_dy \* dy\_dz \* x  **print**(f"Gradiente para pesos: {dL\_dw}") *# Será [0, 0, 0] - no hay aprendizaje* |

Monitoreo neuronas muertas:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **def** dead\_neuron\_stats(activations, threshold=1e-6):  *"""*  *activations: Array de salidas de neuronas (batch\_size, num\_neuronas)*  *returns: Porcentaje de neuronas muertas*  *"""*  batch\_size = activations.shape[0]  dead = np.sum(np.all(activations < threshold, axis=0))  **return** dead / activations.shape[1] \* 100  *# Durante el entrenamiento:*  activations = ... *# Obtener salidas de una capa*  dead\_percent = dead\_neuron\_stats(activations)  **print**(f"Neuronas muertas: {dead\_percent:.2f}%") |

5. Recomendaciones Prácticas

Caso de Uso Función Recomendada

Capas ocultas ReLU (default), GELU, Swish

Visión por computadora Mish, Leaky ReLU

Transformers (BERT, GPT) GELU

Clasificación binaria Sigmoide (capa de salida)

Clasificación multiclase Softmax (capa de salida)

Regresión Identidad (sin activación)

6. Conclusión

Las funciones de activación son clave en el rendimiento de una red neuronal.

ReLU sigue siendo la más usada por su eficiencia.

Funciones modernas (Swish, GELU, Mish) ofrecen mejor rendimiento en casos específicos.

Neuronas muertas pueden mitigarse con Leaky ReLU, inicialización adecuada y ajuste de hiperparámetros.

**🔹 ¿Cuál probar? Depende del problema, pero ReLU es un buen punto de partida.**

**🔹 ¿Experimentar? Prueba GELU en Transformers o Mish en visión por computadora.**

Anexos:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | **import** **numpy** **as** **np**  **import** **matplotlib.pyplot** **as** **plt**  *# Funciones de activación comunes*  **def** sigmoid(x):  **return** 1 / (1 + np.exp(-x))  **def** relu(x):  **return** np.maximum(0, x)  **def** tanh(x):  **return** np.tanh(x)  **def** leaky\_relu(x, alpha=0.01):  **return** np.where(x > 0, x, alpha \* x)  **def** swish(x, beta=1):  **return** x \* sigmoid(beta \* x)  *# Visualización*  x = np.linspace(-5, 5, 100)  plt.figure(figsize=(12, 8))  plt.subplot(2, 3, 1)  plt.plot(x, sigmoid(x))  plt.title("Sigmoid")  plt.subplot(2, 3, 2)  plt.plot(x, relu(x))  plt.title("ReLU")  plt.subplot(2, 3, 3)  plt.plot(x, tanh(x))  plt.title("Tanh")  plt.subplot(2, 3, 4)  plt.plot(x, leaky\_relu(x))  plt.title("Leaky ReLU")  plt.subplot(2, 3, 5)  plt.plot(x, swish(x))  plt.title("Swish")  plt.tight\_layout()  plt.show() |

**Aporte del autor:**

Este informe ofrece una base clara para elegir la función de activación adecuada, dependiendo del contexto. Para investigadores y practicantes, probar funciones modernas puede marcar una diferencia significativa en tareas complejas. Además, es recomendable combinar buenas funciones de activación con técnicas de normalización e inicialización robusta para maximizar el rendimiento de modelos neuronales avanzados.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Función | Rango | Derivabilidad | Problemas | Usos Comunes |
| Sigmoide | (0, 1) | Sí | Vanishing Gradient | Clasificación Binaria |
| Tanh | (-1, 1) | Sí | Vanishing Gradient | Capas ocultas |
| ReLU | [0, ∞) | Sí, a trozos | Neuronas muertas | Visión por computadora |
| Leaky ReLU | [αx, ∞) | Sí | α fijo | Visión, Audio |
| Swish | Variable | Sí | Costosa | Redes profundas |
| GELU | Variable | Sí | Compleja | Transformers |