Aktivierungsfunktionen für neuronale Netze

1. Lineare Aktivierungsfunktion

Diese Funktion gibt den Eingangswert unverändert zurück:

$$f(x) = x$$

Verwendung: Wird selten genutzt, da sie keine Nichtlinearität einführt (außer bei der Ausgabe zB bei einem Regressionsproblem).

2. Stufenfunktion (Step Function)

Eine binäre Aktivierungsfunktion:

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Verwendung: In **Perzeptrons**, aber nicht in modernen neuronalen Netzen.

3. Sigmoid-Funktion

Eine **nichtlineare** Funktion, die Werte zwischen **0 und 1** ausgibt:

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

Eigenschaften:

- Konvertiert große Werte in **1**, kleine Werte in **0**.
- Leidet unter dem **Vanishing Gradient Problem**.

Verwendung: Selten genutzt, da ReLU meist besser ist.

4. Hyperbolische Tangens (Tanh)

Eine verbesserte Version der Sigmoid-Funktion:

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

Eigenschaften:

- Wertebereich: **−1 bis 1**.
- Besser als **Sigmoid**, aber hat ebenfalls das **Vanishing Gradient Problem**.

Verwendung: Wurde in der Vergangenheit oft genutzt, heute eher ReLU.

---

5. ReLU (Rectified Linear Unit)

Die meistgenutzte Aktivierungsfunktion in modernen neuronalen Netzen:

$$f(x) = \max(0, x)$$

Eigenschaften:

- Führt **Nichtlinearität** ein.
- Kein **Vanishing Gradient Problem** für positive Werte.
- Leidet unter dem **Dying ReLU Problem** (Neuronen mit $x < 0$ lernen nichts mehr).

Verwendung: In fast allen modernen neuronalen Netzen.

6. Leaky ReLU

Eine verbesserte Version von ReLU:

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

Eigenschaften:

- **Kleine negative Werte** für $x < 0$, sodass Neuronen nicht komplett “sterben”.
- Standardwert für α ist oft **0.01**.

Verwendung: Alternative zu ReLU, um das **Dying-ReLU-Problem** zu vermeiden.

7. Parametric ReLU (PReLU)

Eine weiterentwickelte ReLU-Version mit **lernbarem** α:

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0, \quad \text{wobei } \alpha \text{ trainierbar ist} \end{cases}$$

Verwendung: In tiefen Netzwerken zur Vermeidung von Dying ReLU.

8. Exponential Linear Unit (ELU)

Eine weitere ReLU-Alternative:

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

Eigenschaften:

- Kein “totes” Neuron,
- Besser als ReLU in manchen Fällen, aber rechenintensiver.

Verwendung: In einigen tiefen Netzen als Ersatz für ReLU.

9. Swish (von Google entwickelt)

Eine **sigmoid-gesteuerte Version von ReLU**:

$$f(x) = x \cdot \text{sigmoid}(x) = x \cdot \frac{1}{1 + e^{-x}}$$

Eigenschaften:

- Weicher als ReLU.
- In manchen Fällen bessere Konvergenz als ReLU.

Verwendung: In neuen Modellen wie **EfficientNet** (Deep-Learning-Modelle für Bildklassifikation).

10. Softmax (für Mehrklassen-Klassifikation)

Wandelt beliebige Werte in **Wahrscheinlichkeiten** um:

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Verwendung: In **Ausgabeschichten** für **Mehrklassenklassifikation**.

Wann wird welche Aktivierungsfunktion verwendet?

Funktion	Wertebereich	Probleme	Verwendung
Step	0 oder 1	Nicht differenzierbar	Perzeptron, nicht in Deep Learning
Sigmoid	$(0, 1)$	Vanishing Gradient	Ältere Netzwerke, selten genutzt
Tanh	$(-1, 1)$	Vanishing Gradient	Ältere Netzwerke, verbessert Sigmoid
ReLU	$[0, \infty)$	Dying ReLU	Standard in CNNs & DNNs
Leaky ReLU	$(-\infty, \infty)$	Keine toten Neuronen	Alternative zu ReLU
PReLU	$(-\infty, \infty)$	Rechenaufwand, ggf. ohne Verbesserung	Erweiterte ReLU-Version
ELU	$(-\infty, \infty)$	Aufwändiger als ReLU	Alternative zu ReLU

Funktion	Wertebereich	Probleme	Verwendung
Swish	$(-\infty, \infty)$	Rechenaufwand	In modernen Architekturen (Google)
Softmax	$(0, 1)$, Summe = 1	-	Letzte Schicht für Mehrklassenklassifikation

Häufigste Wahl:

- **ReLU** für versteckte Schichten.
- **Softmax** für Mehrklassen-Klassifikation.
- **Sigmoid** für Binärklassifikation (heute meist durch ReLU ersetzt).

Problemerklärung:

- **Nicht differenzierbar:**
  Neuronale Netze werden mit dem Gradientenabstieg trainiert, der die Ableitung (Gradienten) der Kostenfunktion benötigt, um Gewichte anzupassen.
  Wenn eine Aktivierungsfunktion **nicht differenzierbar** ist, kann **Backpropagation** an dieser Stelle nicht korrekt arbeiten → Das Lernen stoppt oder wird instabil.

- **Vanishing Gradient:**
  Das **Vanishing Gradient Problem** tritt auf, wenn die **Gradienten (Ableitungen)** in einem neuronalen Netz während des Trainings **zu klein** werden, sodass die **Gewichtsaktualisierung stoppt** oder extrem langsam wird.

- **Dying ReLU:**
  Das **Dying ReLU Problem** tritt auf, wenn zu viele Neuronen in einem neuronalen Netz immer **0 ausgeben** und damit **nicht mehr lernen können**.

- **Keine toten Neuronen:**
  Das bedeutet, dass für einige Neuronen die Ausgabe immer **0** bleibt, unabhängig von der Eingabe.
  Falls die Gewichte so angepasst werden, dass ein Neuron **immer negatives** x **erhält**, bleibt seine Ausgabe für immer **0** → **es stirbt** und lernt nichts mehr.