#srm #week3

WEEK 3 - DAY 4

AUG-19-2025 (8:00AM - 9:40AM IST)

10:30 PM - 12:40 AM ET

ACTIVATION FUNCTIONS MASTERY

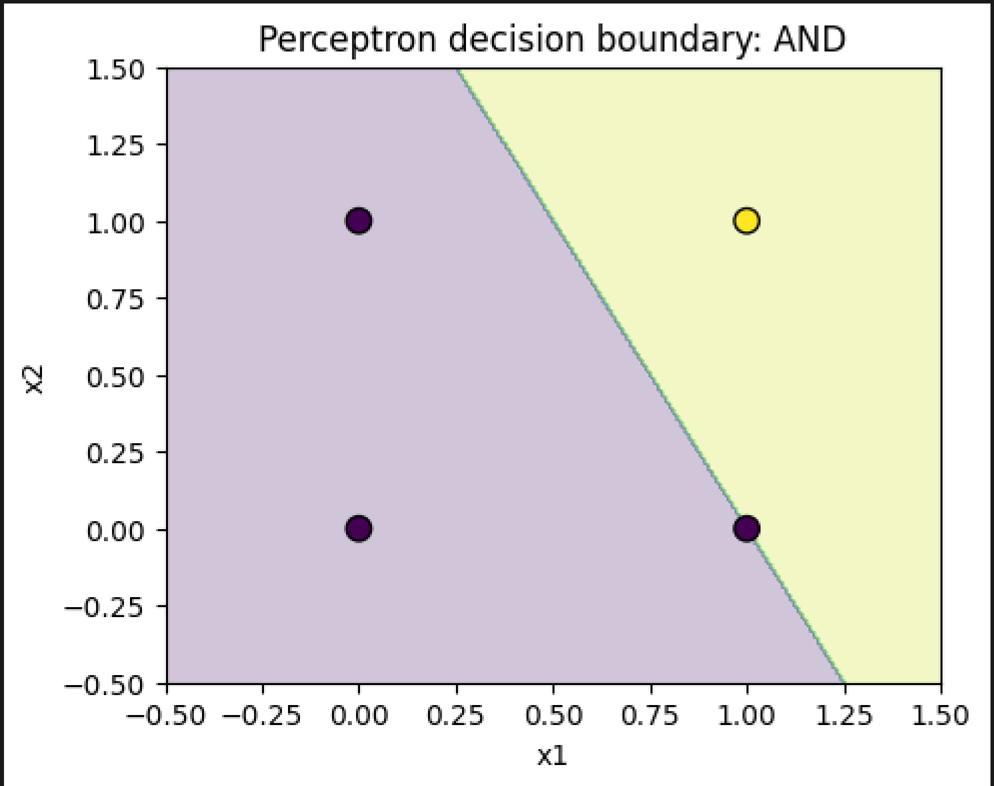
- **Sigmoid, Tanh, ReLU Family** - Mathematical properties, gradients, use cases
- **Advanced Activations** - Leaky ReLU, ELU, Swish, GELU
- **Practical Selection Criteria** - When and why to choose specific functions

WEEK 1

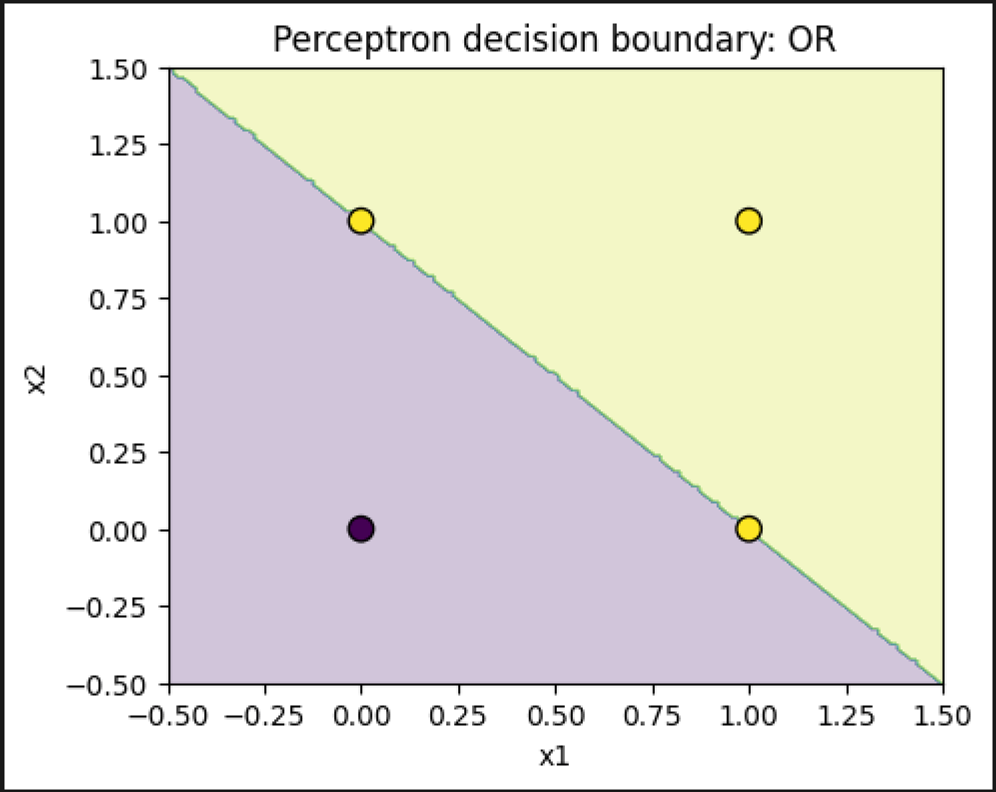
Approach	How it Works	Example
Rule-Based	Human writes explicit rules	"If temperature > 30°C, recommend shorts"
Traditional ML	Human defines features, algorithm finds patterns	"Extract 20 weather features, train decision tree"
Deep Learning	Algorithm learns features AND patterns	"Give raw weather data, predict clothing"

WEEK 2

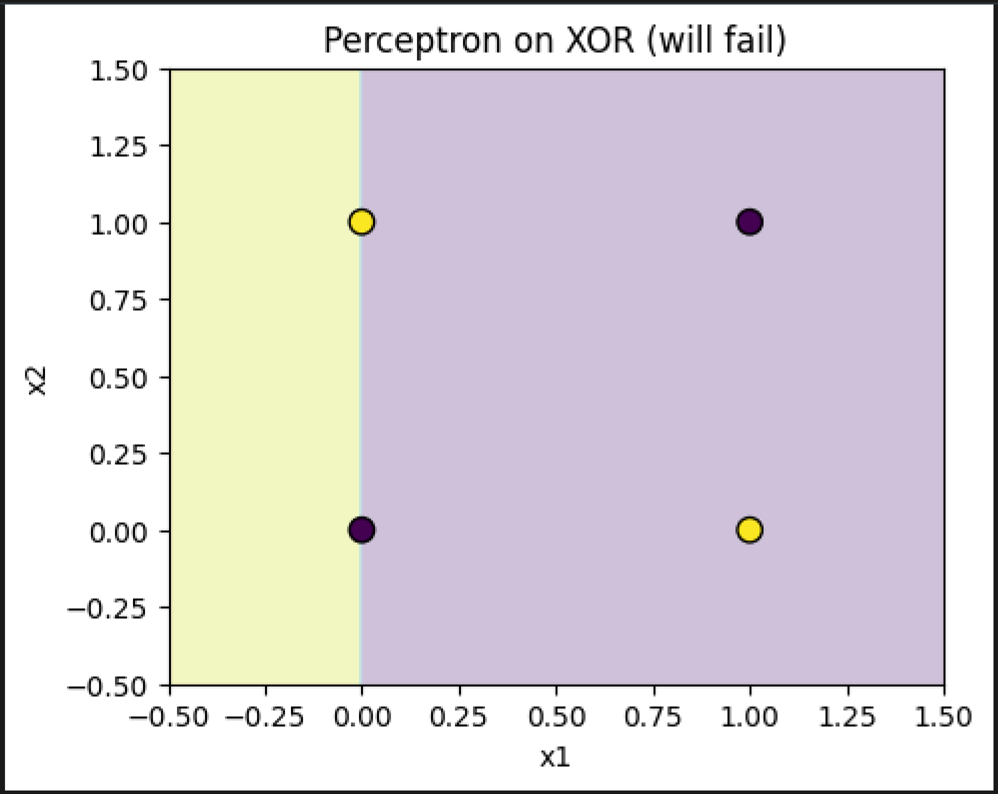
AND PREDICTIONS: [0 0 0 1]



OR PREDICTIONS : [0 1 1 1]



CORRECT LABELS : [0 1 1 0]

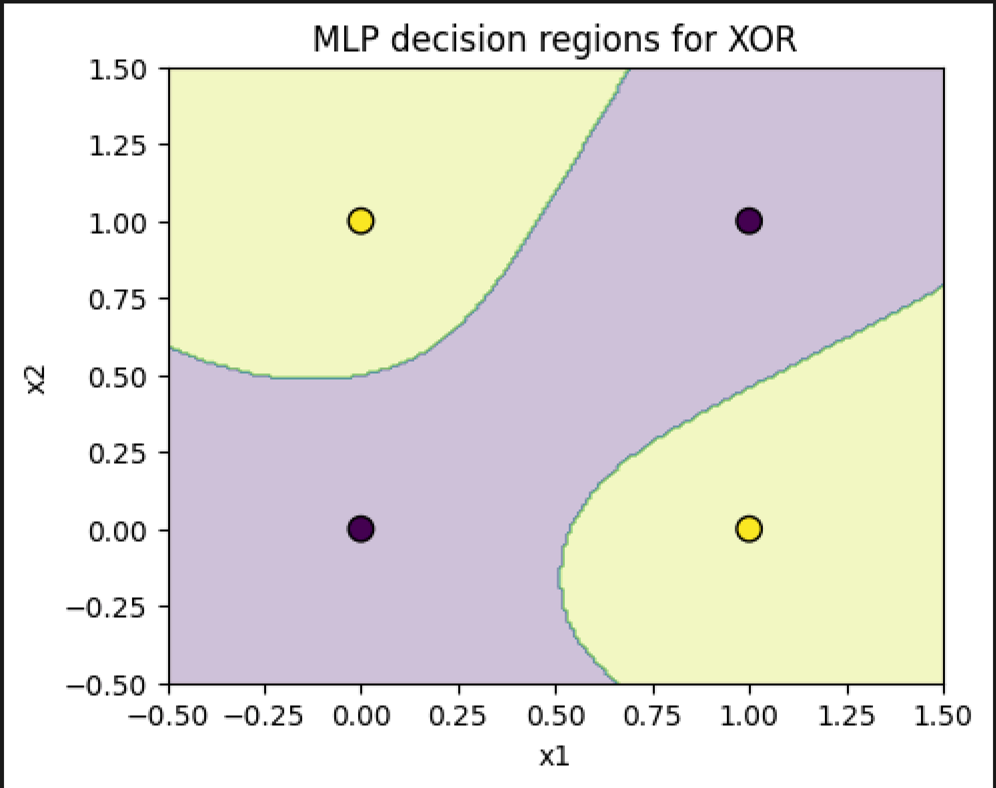


HERE HOW WE SOLVED IT

```python
model = Sequential([
    Dense(4, activation='tanh', input_shape=(2,)),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.1),
              loss='binary_crossentropy', metrics=['accuracy'])
hist = model.fit(X, y, epochs=500, verbose=0)
print('Final accuracy on XOR:', model.evaluate(X, y, verbose=0)[1])
print('Predictions:', (model.predict(X) > 0.5).astype(int).ravel())
```

GRAPH

```
Final accuracy on XOR: 1.0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step
Predictions: [0 1 1 0]
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 1s 952us/step
```



WEEK 3 THIS WEEK - ACTIVATION

```python
# Create model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(784,)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

Step 1: Recall the rule

A **dense layer** takes an input vector x, applies weights W, adds a bias b, and passes it through an activation function f:

$$y = f(Wx + b)$$

- x = input vector (features)
- W = weight matrix (how strongly each input connects to each output)
- b = bias vector (shift)
- $f(\cdot)$ = activation function (ReLU, Swish, etc.)

EXAMPLES

- Restaurant - everyone eats same food , but people pay different amout -> tips => bias
- Factor : Fine tunning the instruments appling wight
  - Raw material = x input
- f -> activation function

WHAT IS ACTIVATION

🔑 **Activations (the "gatekeepers" in a neural net)**

1. **ReLU (Rectified Linear Unit)**
   - Rule: pass positive values, block negatives (set them to 0).
   - Think: a **light switch** — off below 0, on above 0.
2. **Leaky ReLU**
   - Rule: same as ReLU, but negatives are not killed — they leak a little.
   - Think: a **safety valve** — lets a trickle of negative flow.
3. **Swish**
   - Rule: multiply input by a smooth sigmoid → negatives shrink but don't vanish.
   - Think: an **auto-dimmer** — dims weak signals smoothly.
4. **GELU (Gaussian Error Linear Unit)**
   - Rule: input gets passed depending on probability (via Gaussian curve).
   - Think: a **confidence gate** — only strong signals get fully through.

WHAT IS DERIVATIVES

📉 **Derivatives (how much the function "pushes" during learning)**

1. **ReLU derivative**
   - 0 for x < 0 → dead neurons possible.
   - 1 for x > 0 → strong, stable gradient.
2. **Leaky ReLU derivative**
   - Small slope (e.g. 0.1) when x < 0 → prevents dead neurons.
   - Slope = 1 when x > 0.
3. **Swish derivative**
   - Never flat zero → always some gradient.
   - Smoother changes help gradients flow better in deep nets.
4. **GELU derivative**
   - Curved like a Gaussian → soft, probabilistic slope.
   - Keeps gradients alive while tapering extremes.

WHAT IS GRADIENT

- It is slope in the diagram
- more an more, slowly the voice will come down , neurons will not learn

REAL WORLD ANALOGIES

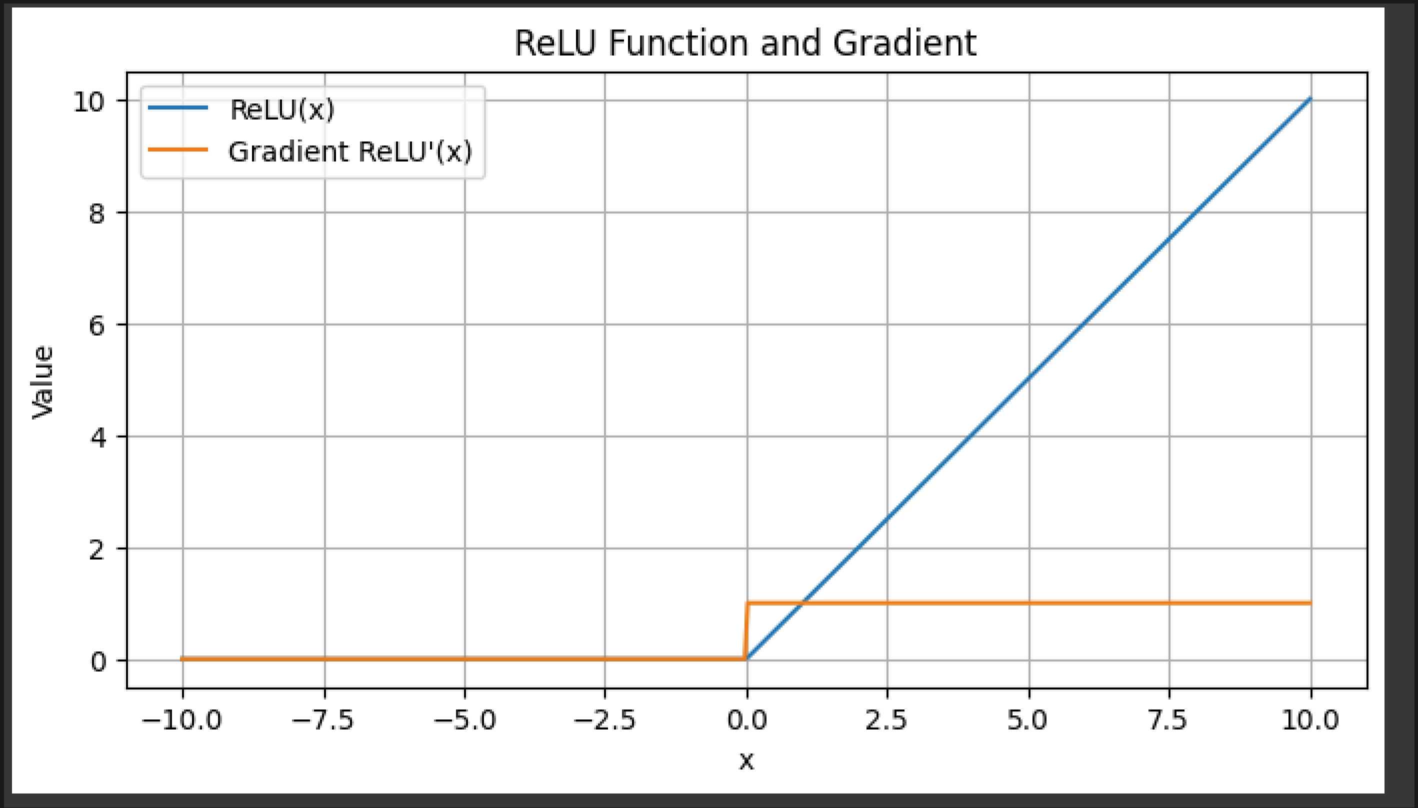
1. **ReLU**
    - **Activation:** Like an **automatic door** that only opens if you push forward (positive). If you push backwards (negative), it stays shut.
    - **Derivative:** Once the door is closed ($x < 0$), no matter how hard you push, it doesn't move (slope = 0). When it's open ($x > 0$), it moves freely (slope = 1).
    - ⚠️ Risk: some doors get stuck permanently closed ("dead neurons").

- Water going through pipe, open , +ve full force
- Closed - blocked Stoped

FORMULA

ReLU	$f(x) = \max(0, x)$	$[0, \infty)$	$f'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$

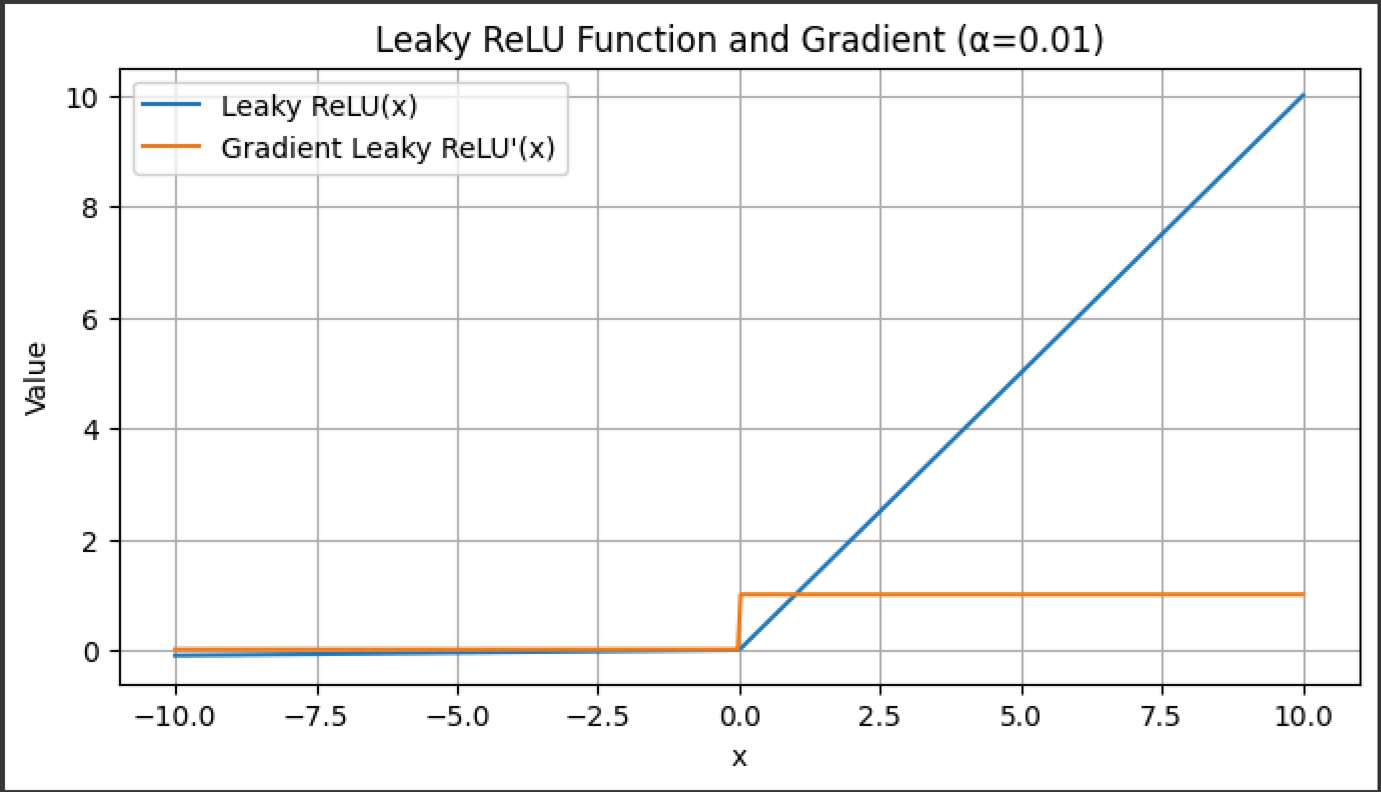
$-$No vanishing gradient for $x > 0$ $-$Sparse activation (efficient) $-$Fast computation	$-$Dying ReLU (neurons stuck at 0) $-$Not zero-centered $-$Undefined at $x = 0$



2. **Leaky ReLU**
    - **Activation:** Same door, but with a **small side vent** — even if you push backwards, a little airflow comes through.
    - **Derivative:** That tiny airflow means the gradient never completely dies — there's always some signal to adjust weights.
    - ✅ Fixes the "dead door" problem.

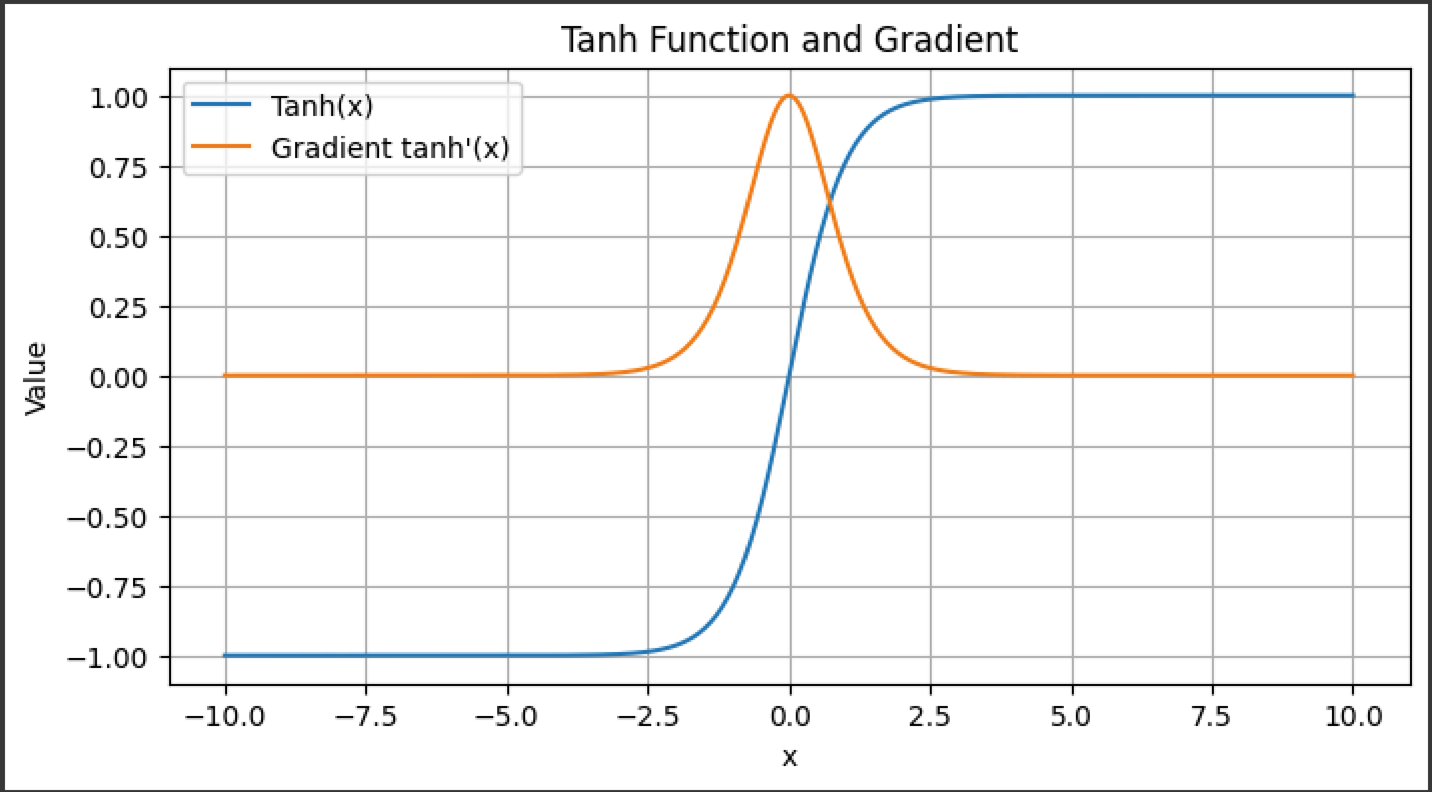
Leaky ReLU / Variants	$f(x) = \begin{cases} x & x > 0 \\ \alpha x & x \le 0 \end{cases}, \alpha \approx 0.01$	$(-\infty, \infty)$	$f'(x) = \begin{cases} 1 & x > 0 \\ \alpha & x \le 0 \end{cases}$

–Fixes dying ReLU issue –Maintains ReLU speed –Variants (PReLU, ELU) smoother	–Leak α needs tuning –Still not fully symmetric –ELU adds extra computation



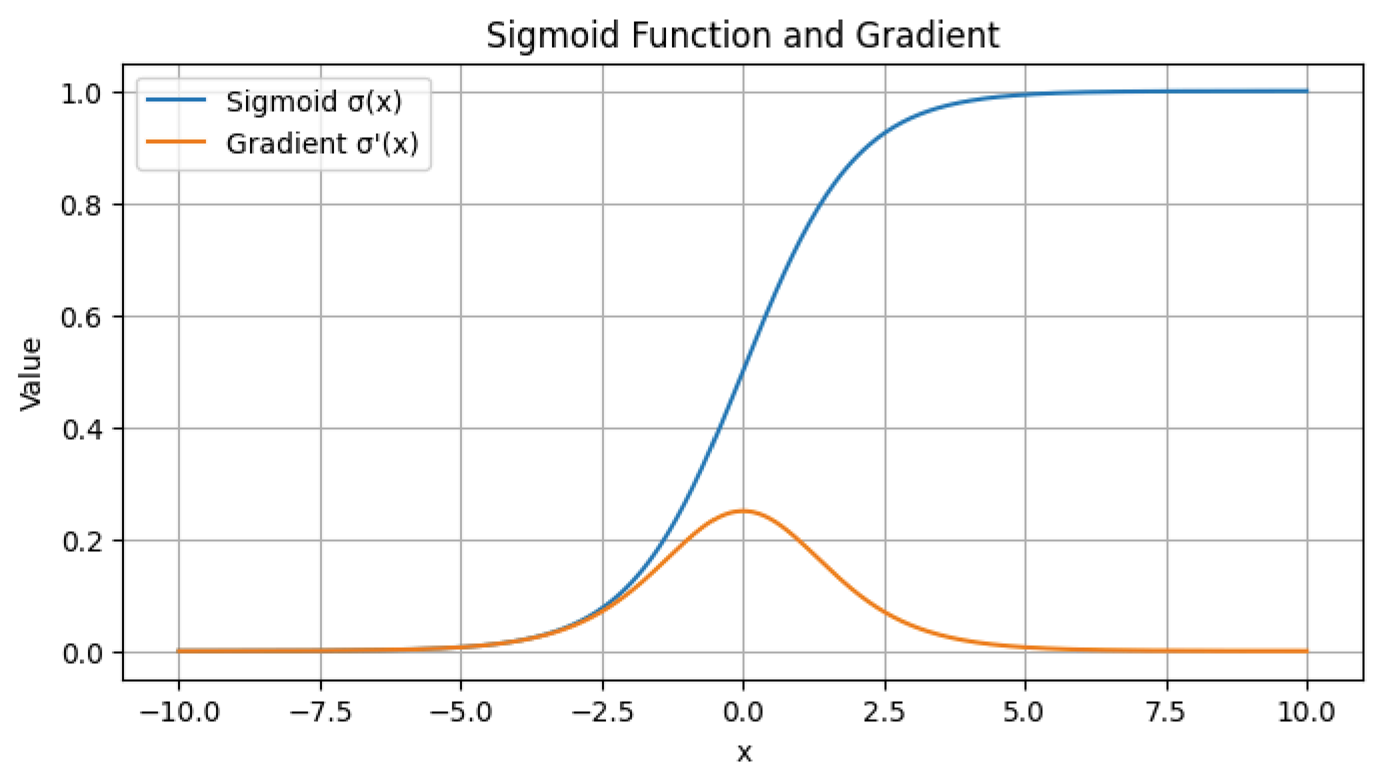
TANH

Tanh	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$[-1, 1]$	$\tanh'(x) = 1 - \tanh^2(x) \quad (\max: 1 \text{ at } x = 0)$

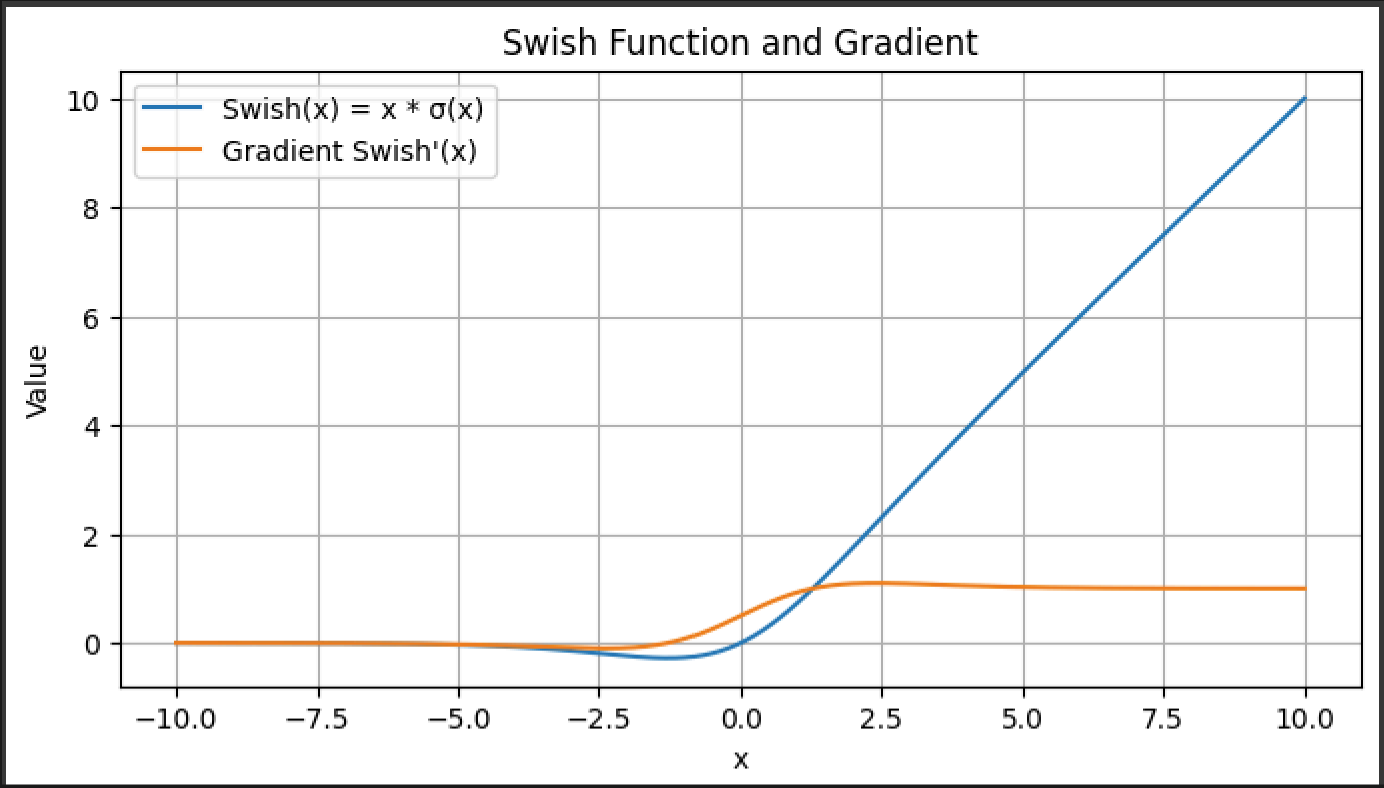


5. **Sigmoid (classic, for contrast)**
    - **Activation:** Like a **saturation dial** on a photo editor. Small values adjust brightness, but after a point the image looks unchanged.
    - **Derivative:** Once saturated (very dark or very bright), no matter how much you turn, nothing changes (gradient ≈ 0).
    - ⚠️ Classic cause of vanishing gradients.

Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$[0, 1]$	$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \quad (\text{max} : 0.25 \text{ at } x = 0)$

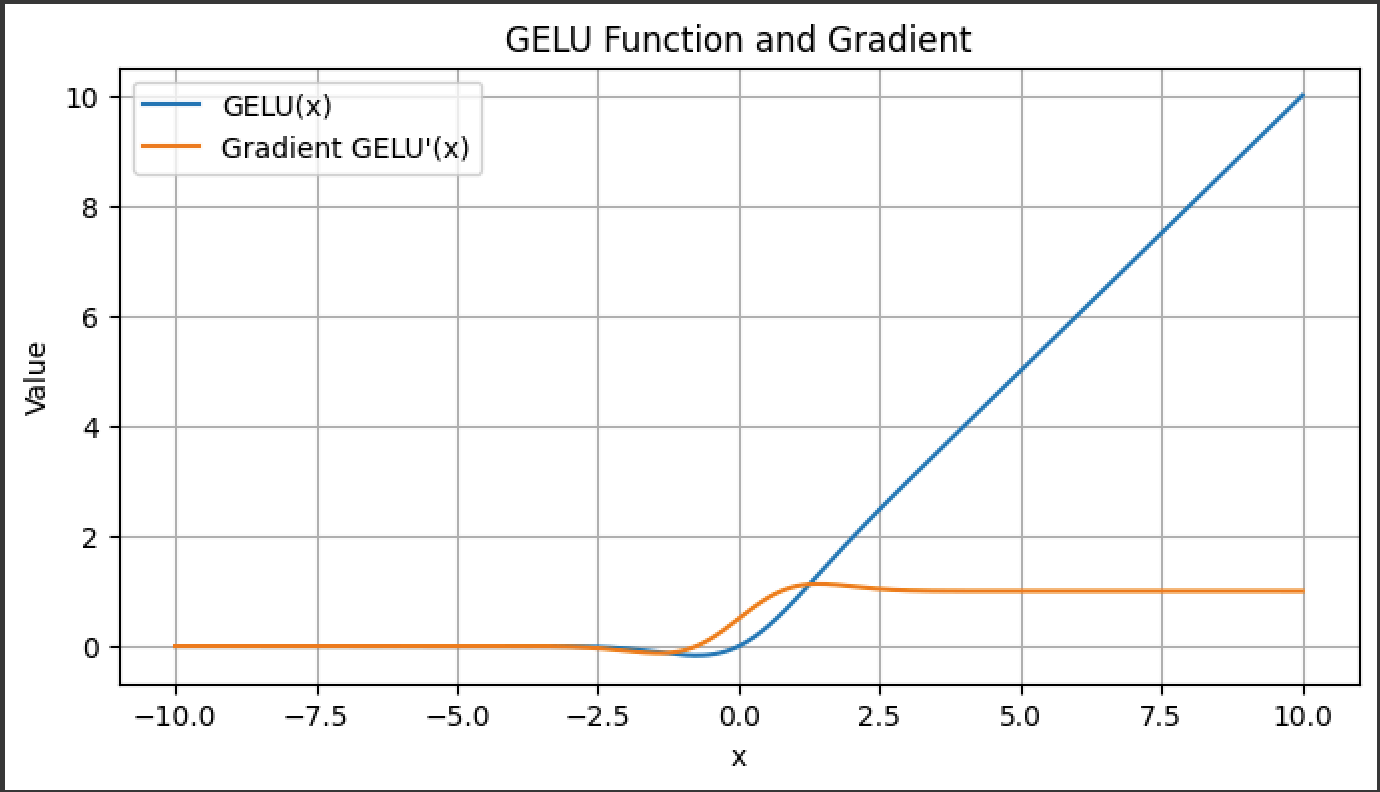

Sigmoid Function and Gradient
Sigmoid σ(x)
Gradient σ'(x)
Value
1.0
0.8
0.6
0.4
0.2
0.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

3. **Swish**
    - **Activation:** Think of a **water tap with a smooth knob**. Small pushes give a trickle, bigger pushes give a steady stream — no sudden ON/OFF.
    - **Derivative:** Since flow changes smoothly, the adjustment (gradient) is never flat zero. Training feels “smoother” — less jerky than ReLU.

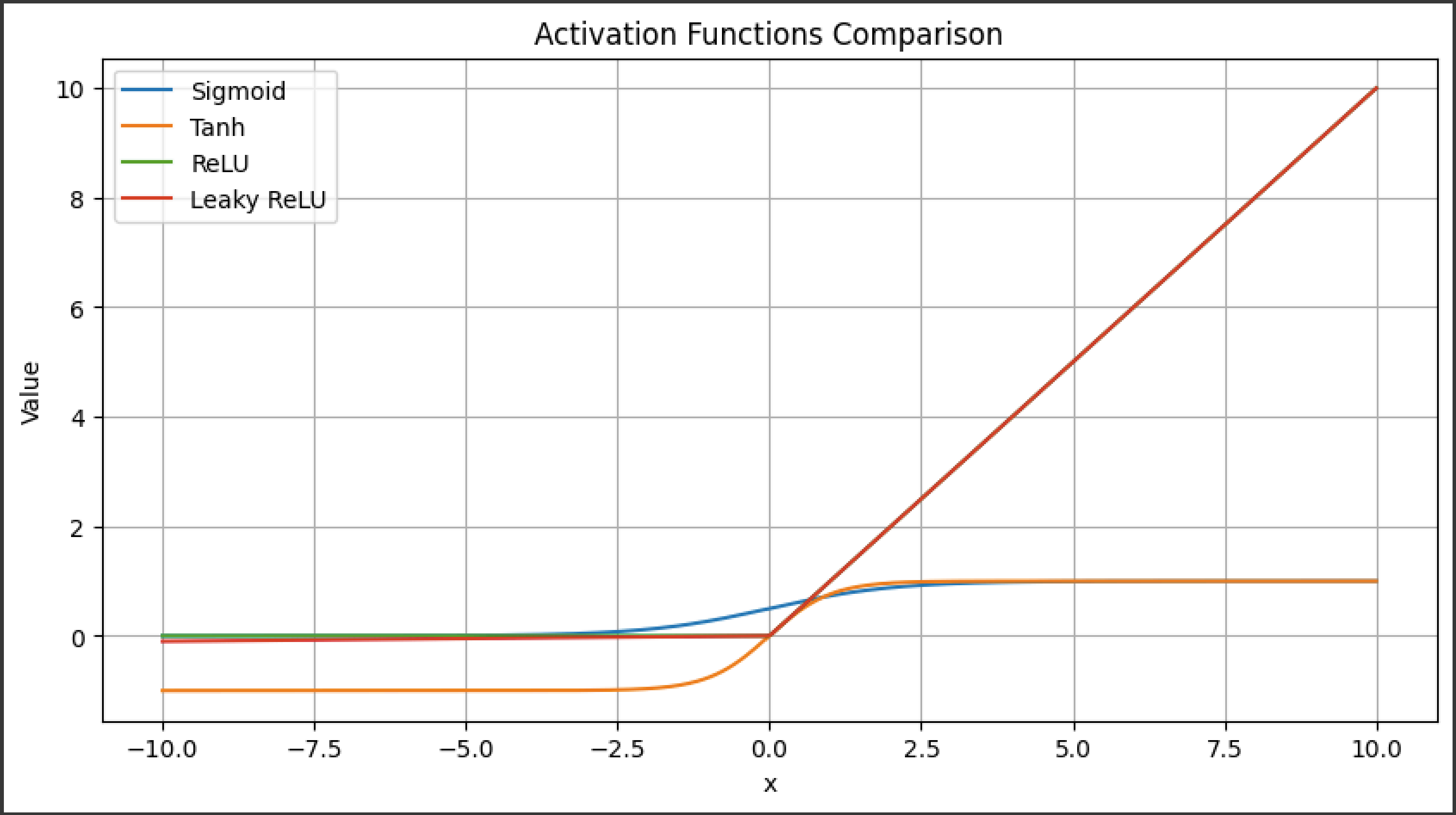

Swish Function and Gradient
Swish(x) = x * σ(x)
Gradient Swish'(x)
Value
x
10
8
6
4
2
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

4. GELU

- **Activation:** Imagine a **smart filter in a call center**. Calls (signals) get through based on how "confident" the filter is that they're important.
- **Derivative:** The filter doesn't fully block weak signals, it just reduces them softly. That keeps some learning signal alive, but prioritizes stronger inputs.



ALL IN ONE



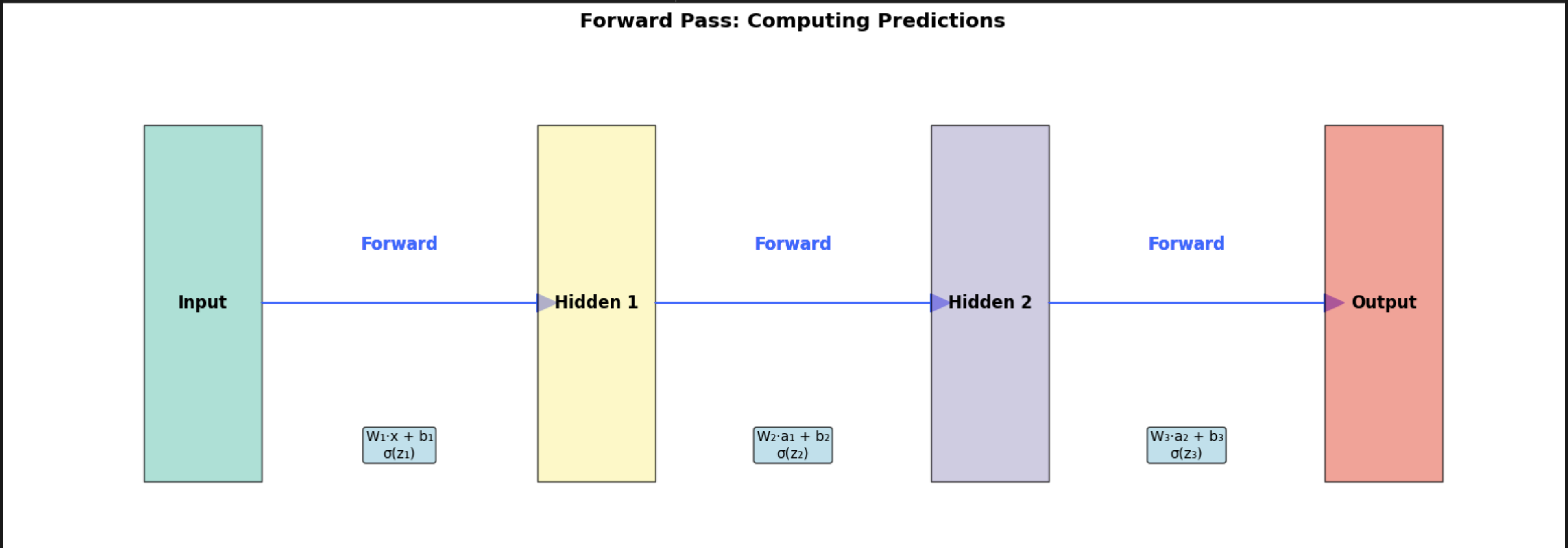
Activation Functions Comparison
Sigmoid
Tanh
ReLU
Leaky ReLU
Value
x
10
8
6
4
2
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

BREAK

RECAP

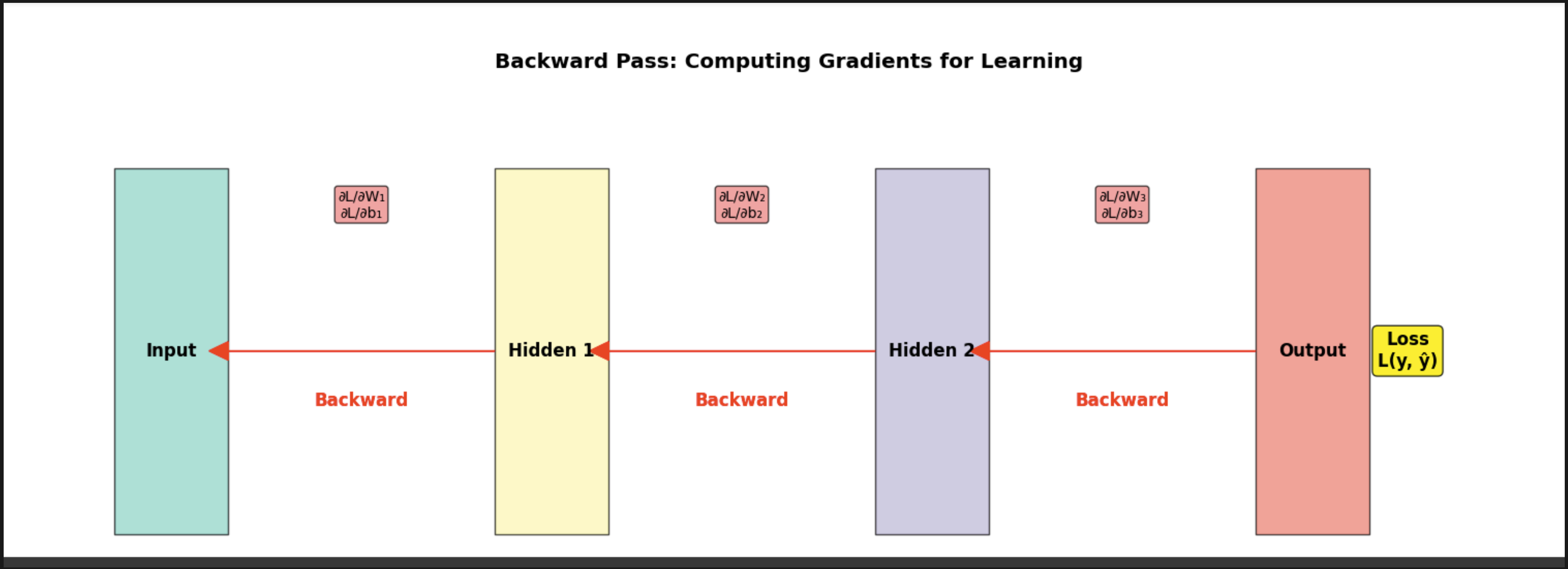
- ReLU -> water flowing on the pipe , closed stped
- Leaky -> avoid dead neurons - 0.02 make it flow
- Tanh -> Extreme weather -1 to +1 centered at 0
- Sigmoid -> Variable light , slow changes, but -ve pay the price
- Swish -> Walking to the automatic door, walkback door still open
- Gelu --> Security clearance , may be may not be caught

FOWARD PASS



Forward Pass: Computing Predictions
Input
Forward
Hidden 1
Forward
Hidden 2
Forward
Output
$W_1 \cdot x + b_1$
$\sigma(z_1)$
$W_2 \cdot a_1 + b_2$
$\sigma(z_2)$
$W_3 \cdot a_2 + b_3$
$\sigma(z_3)$

BACKWARD PASS



APPLY MANUALLY

Step 1: Recall the rule

A **dense layer** takes an input vector x, applies weights W, adds a bias b, and passes it through an activation function f:

$$y = f(Wx + b)$$

- x = input vector (features)
- W = weight matrix (how strongly each input connects to each output)
- b = bias vector (shift)
- $f(\cdot)$ = activation function (ReLU, Swish, etc.)

Step 2: Set up a toy example

Say we have:

- Input size = 2
- Output size = 3

  So:
- $x \in \mathbb{R}^2$
- $W \in \mathbb{R}^{3\times 2}$
- $b \in \mathbb{R}^3$

Let's pick some numbers:

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad W = \begin{bmatrix} 1 & -1 \\ 0 & 2 \\ 3 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}$$

Step 3: Multiply Wx

$$Wx = \begin{bmatrix} 1 & -1 \\ 0 & 2 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} (1)(1) + (-1)(2) \\ (0)(1) + (2)(2) \\ (3)(1) + (1)(2) \end{bmatrix} = \begin{bmatrix} -1 \\ 4 \\ 5 \end{bmatrix}$$

So the multiplication:

$$Wx = \begin{bmatrix} 1 & -1 \\ 0 & 2 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 4 \\ 5 \end{bmatrix}$$

is **valid** because the **inner dimensions match**: $(3 \times 2) \cdot (2 \times 1)$.

If you try xW:

$$xW = \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 2 \\ 3 & 1 \end{bmatrix}$$

- Here x is (1×2) and W is (3×2).
- The **inner dimensions (2 and 3) do not match** → ❌ multiplication is **not possible.**

✅ So $Wx \neq xW$.

Only Wx works in this example.

Step 4: Add the bias b

$$Wx + b = \begin{bmatrix} -1 \\ 4 \\ 5 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ 5 \\ 4 \end{bmatrix}$$

Step 5: Apply activation function f

Suppose f is ReLU (max(0, x)):

$$y = f(Wx + b) = \max(0, \begin{bmatrix} -1 \\ 5 \\ 4 \end{bmatrix}) = \begin{bmatrix} 0 \\ 5 \\ 4 \end{bmatrix}$$

✅ Final output:

$$y = \begin{bmatrix} 0 \\ 5 \\ 4 \end{bmatrix}$$

Why this matters for students

- **Shapes**: every time, check dimensions match (W is output × input).
- **Meaning**:
  - Multiply = “mix features with weights”
  - Bias = “shift/calibrate”
  - Activation = “decide what flows forward”

HOME WORK

Problem A

Input size = 2, Output size = 2

$$x = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \quad W = \begin{bmatrix} 1 & 3 \\ -2 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Problem B

Input size = 3, Output size = 2

$$x = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}, \quad W = \begin{bmatrix} 2 & -1 & 0 \\ 1 & 3 & -2 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Problem C

Input size = 2, Output size = 3

$$x = \begin{bmatrix} -1 \\ 4 \end{bmatrix}, \quad W = \begin{bmatrix} 0 & 2 \\ 3 & -1 \\ 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix}$$

BOOKS REFERENCE

Week-3 Sessions 7,8,9

Book	Relevant Chapters	Key Concepts Covered	Best Use in Session	Limitations	
Aggarwal (2018)	Ch. 1, 2, 3	Neurons, XOR, sigmoid/tanh/ReLU, gradients, vanishing gradients, Universal Approx.	Theoretical depth, derivations, XOR hook	Swish/GELU may be missing	
Goodfellow et al. (2017)	Ch. 6, 8	MLPs, all activations, gradients, Universal Approx., vanishing/dying ReLU	Core theory, rigorous math, gradient issues	Swish/GELU likely absent	
Chollet (2018)	Ch. 2, 4	Sigmoid/tanh/ReLU, Python demos, applications (spam, sentiment)	Python plotting, practical bridges	Light on derivations, no Swish/GELU	
Kim (2017)	Ch. 2, 3	Perceptrons, XOR, sigmoid/tanh/ReLU, gradients	XOR and classical activations (Matlab)	Matlab focus, no Swish/GELU	
Venkatesan & Li (2018)	Ch. 2, 3/4	ReLU in CNNs, gradients, vision applications	ReLU vision example	CNN-focused, limited XOR/MLPs	
Manaswi (2018)	Ch. 2, 3	XOR, sigmoid/tanh/ReLU, Python, applications	Python demos, practical examples	No Swish/GELU, light derivations	

LAST SLIDE

- Podcast mp3
- AI Agent