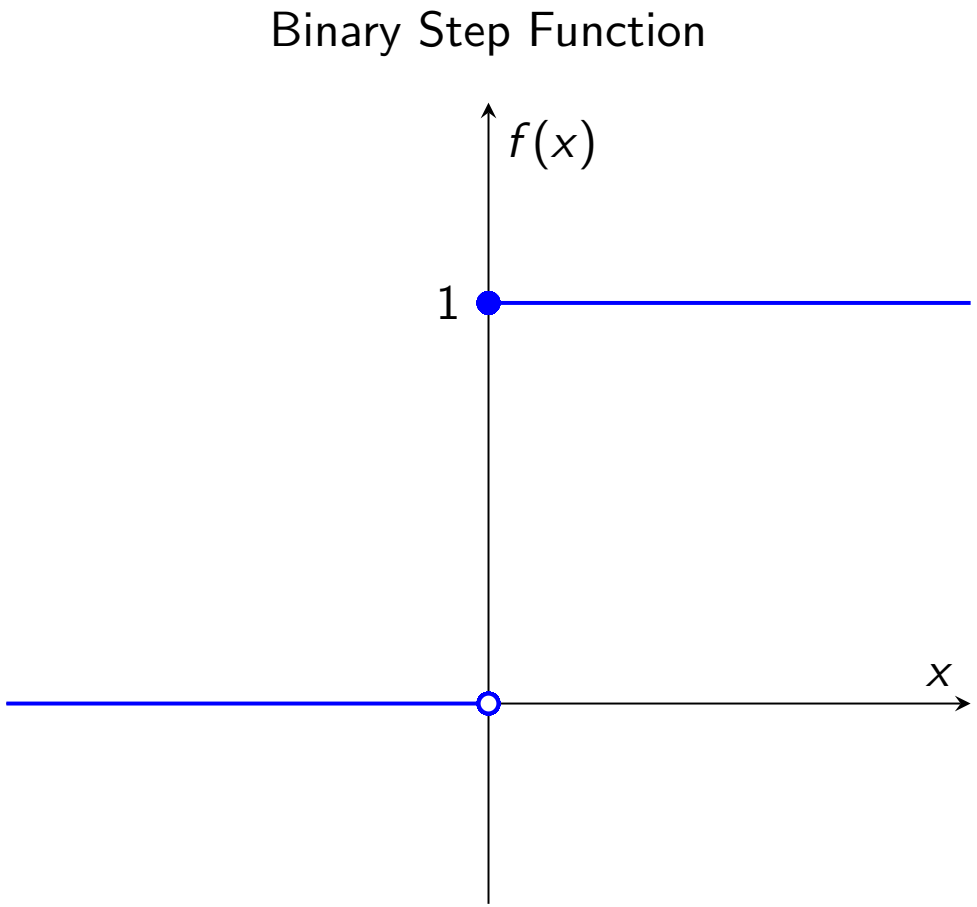
Activation Functions

1. Binary Step Function - Figure



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

1. Binary Step Function - Analysis

Properties

- Output is either 0 or 1 ("fires" or "doesn't fire").
- A non-linear function.

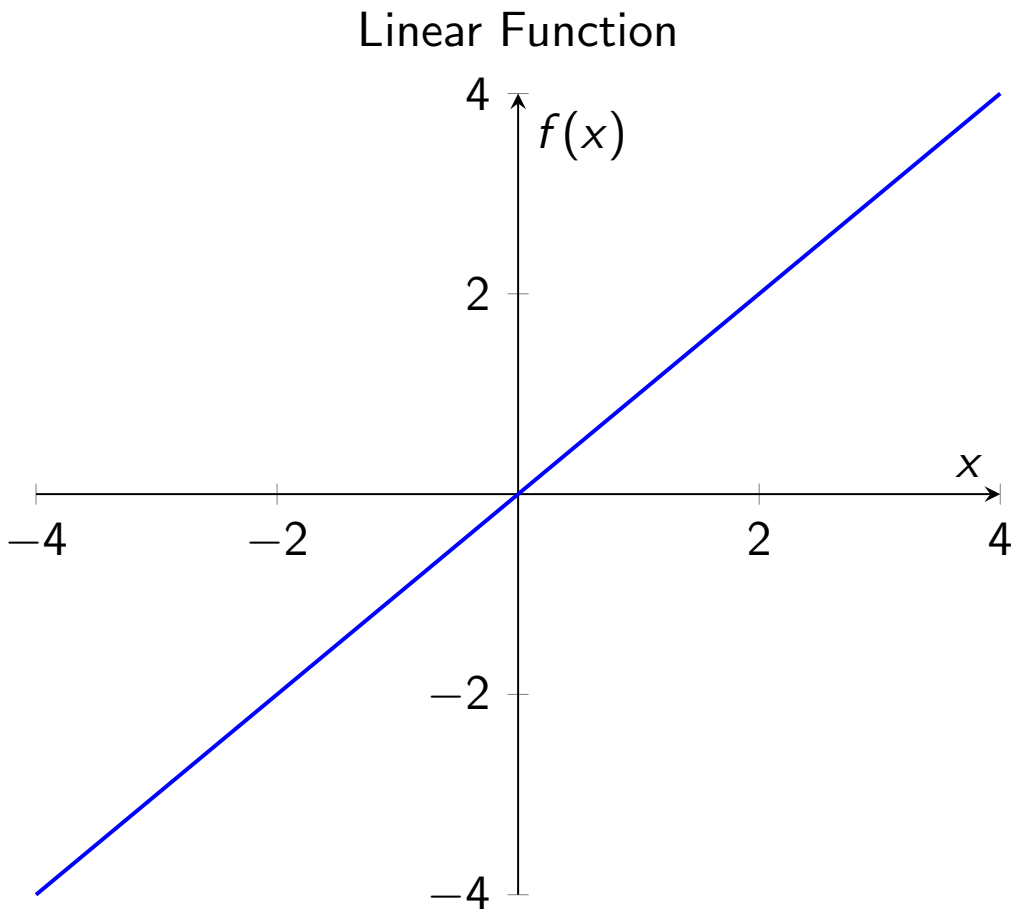
Drawbacks

- **Zero Gradient:** The derivative is 0 everywhere except at $x = 0$ (where it's undefined).
- This means **backpropagation cannot work** as no gradient flows back to update weights.
- Cannot be used for multi-class classification.

Use

- Rarely used in modern networks.
- Historically important (e.g., in the original **Perceptron**).

2. Linear Function - Figure



$$f(x) = x$$

2. Linear Function - Analysis

Properties

Identity function: Output is equal to the input.

Derivative is constant ($f'(x) = 1$).

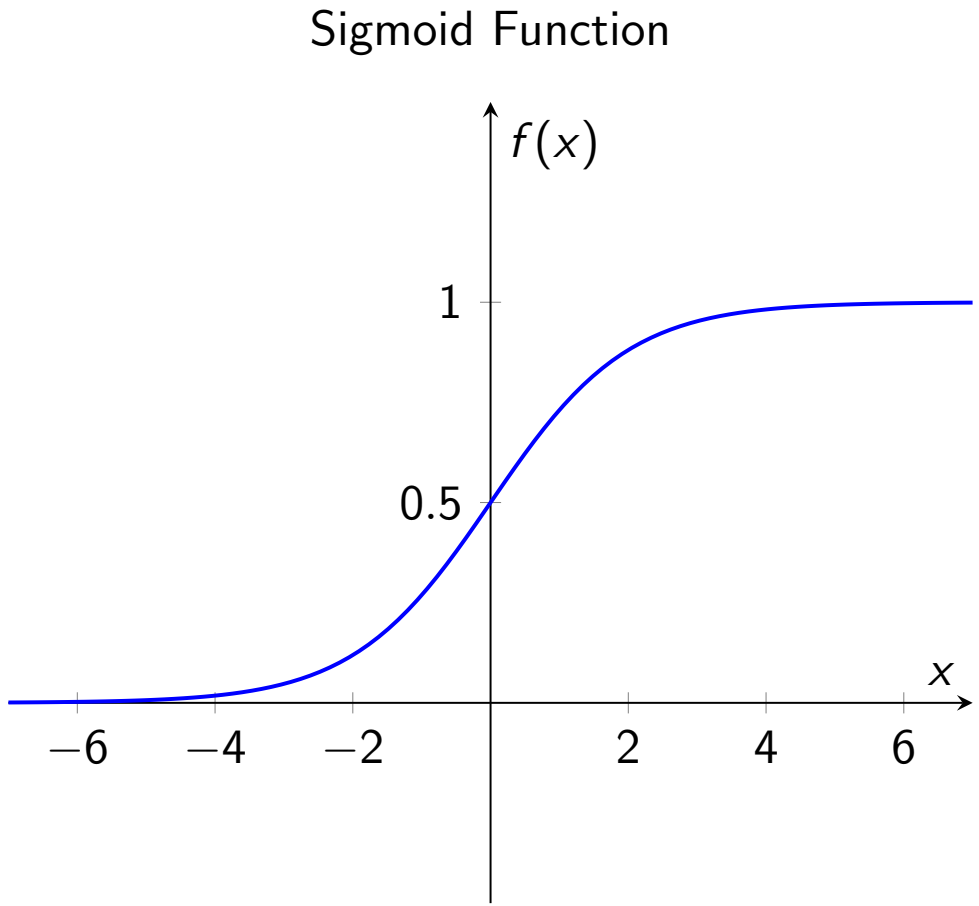
Drawbacks

- **No Non-linearity:** Using this in hidden layers is pointless.
- A stack of linear layers is mathematically equivalent to a **single linear layer**.
- The network cannot learn complex, non-linear patterns.

Use

- Typically only used in the **output layer** for **regression tasks** (where the output value can be any real number).

3. Sigmoid / Logistic - Figure



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

3. Sigmoid / Logistic - Analysis

Properties

- S-shaped curve, **Range:** (0, 1).
- Differentiable and smooth.

Advantages

- Output can be interpreted as a **probability**.
- Smooth gradient.

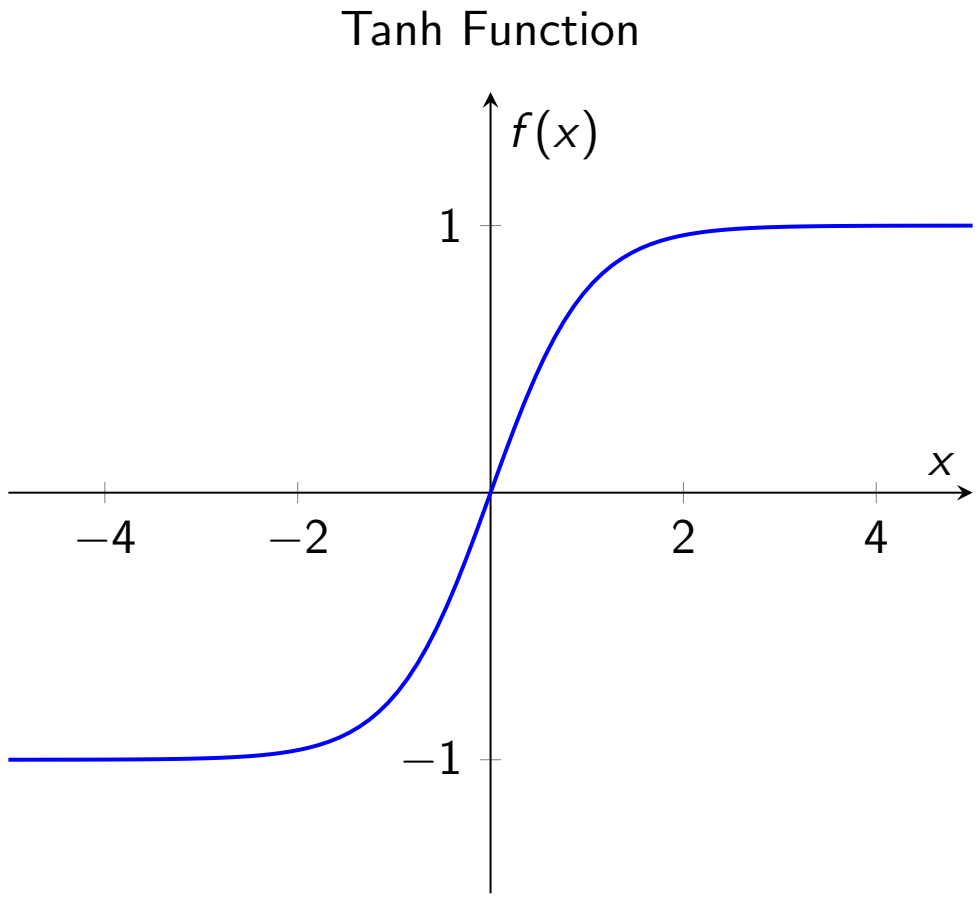
Drawbacks

- **Vanishing Gradients:** For large positive or negative x, the gradient ≈ 0.
- **Not zero-centered:** Output is always positive. This can make training unstable (all gradients for weights in a layer will have the same sign).

Use

- **Output layer** for **binary classification**.
- Rarely used in hidden layers today (Tanh or ReLU are preferred).

4. Tanh (Hyperbolic Tangent) - Figure



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

4. Tanh (Hyperbolic Tangent) - Analysis

Properties

- S-shaped, like a "scaled" sigmoid.
- **Range:** (-1, 1).
- Differentiable and smooth.

Advantages

- **Zero-centered:** Output mean is near 0. This helps center the data for the next layer and makes training easier and more stable than Sigmoid.

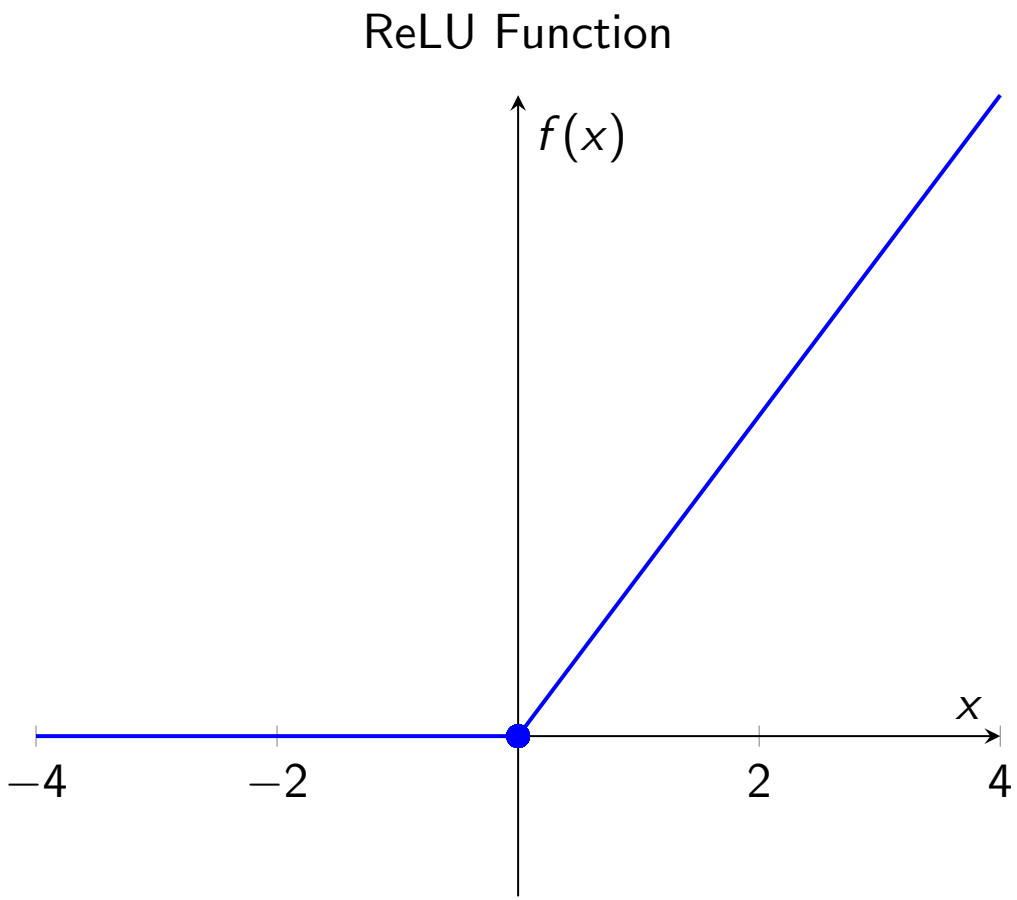
Drawbacks

- Still suffers from the **vanishing gradient problem** at the extremes (though the gradient is steeper than Sigmoid).

Use

- Commonly used in **hidden layers** of standard feed-forward networks.
- Very common in **Recurrent Neural Networks (RNNs)**.
- Generally preferred over Sigmoid for hidden layers.

5. ReLU (Rectified Linear Unit) - Figure



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

5. ReLU (Rectified Linear Unit) - Analysis

Properties

- Linear for $x > 0$, zero for $x \leq 0$.
- **Non-saturating** (for $x > 0$).

Advantages

- **Computationally efficient:** Very simple max operation.
- **Solves vanishing gradient** for positive inputs.
- **Sparsity:** Induces sparsity as many neurons output 0.

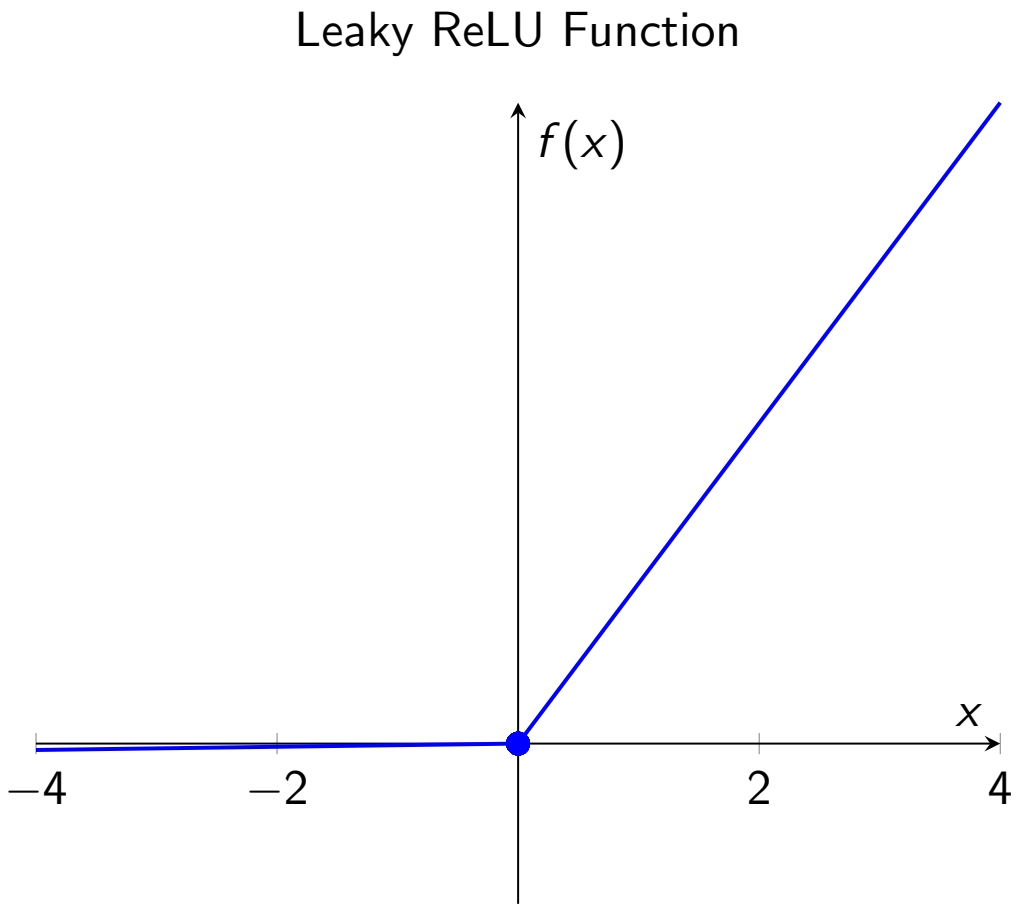
Drawbacks

- **The "Dying ReLU" Problem:** If a neuron's input is always negative (e.g., due to a large negative bias), it gets "stuck" outputting 0. Its gradient will always be 0, and it will never learn again.
- **Not zero-centered**.

Use

- The **most common** activation function for **hidden layers** in **CNNs** and **DNNs**. It is the default choice.

6. Leaky ReLU - Figure



$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.01x & \text{if } x < 0 \end{cases}$$

6. Leaky ReLU - Analysis

Properties

- An improved version of ReLU.
- Introduces a small, non-zero slope for negative inputs (e.g., $\alpha = 0.01$).

Advantages

- **Solves the "Dying ReLU" Problem:** The gradient is never 0, so neurons can always recover and continue learning.
- Retains the efficiency and non-saturation benefits of ReLU.

Drawbacks

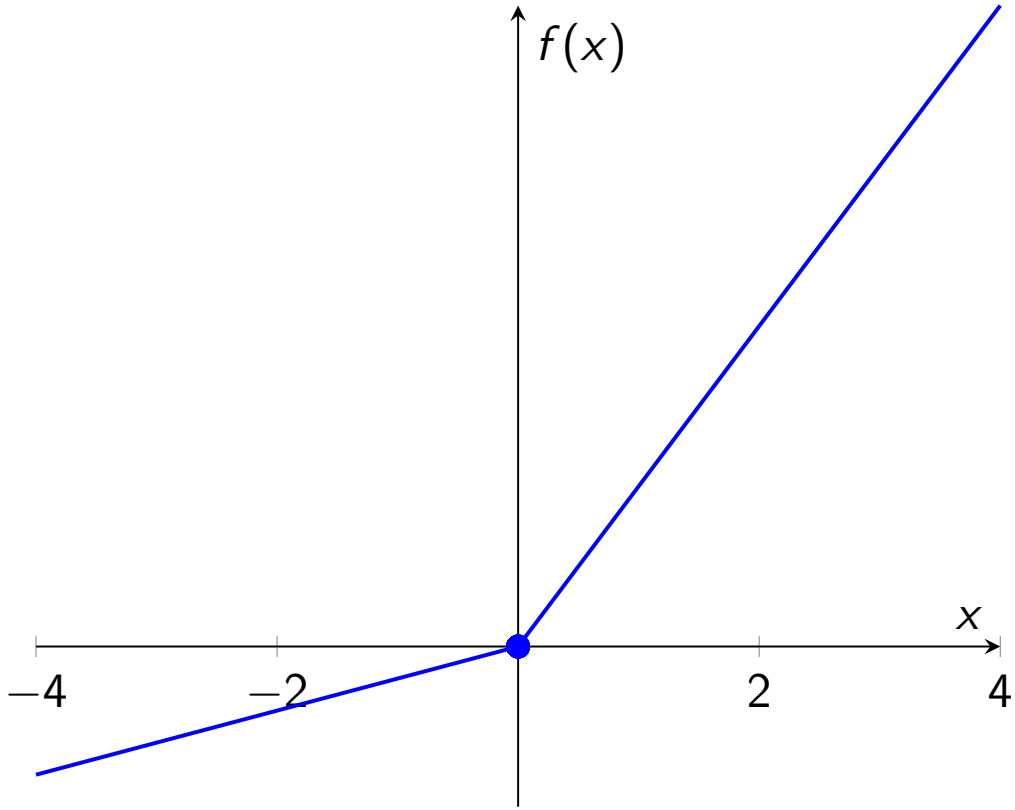
- Results are not always consistent; sometimes it works better than ReLU, sometimes not.
- The slope α (0.01) is a fixed hyperparameter that must be chosen.

Use

- A common alternative to ReLU, especially if "Dying ReLU" is a suspected problem.

7. Parameterized ReLU (PReLU) - Figure

PReLU Function (example $\alpha = 0.2$)



$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$$

7. Parameterized ReLU (PReLU) - Analysis

Properties

- A variation of Leaky ReLU where α is a **learnable parameter**.

Advantages

- The network **learns the best slope** α during backpropagation.
- Can adapt to the data better than Leaky ReLU.
- Solves the "Dying ReLU" problem.

Drawbacks

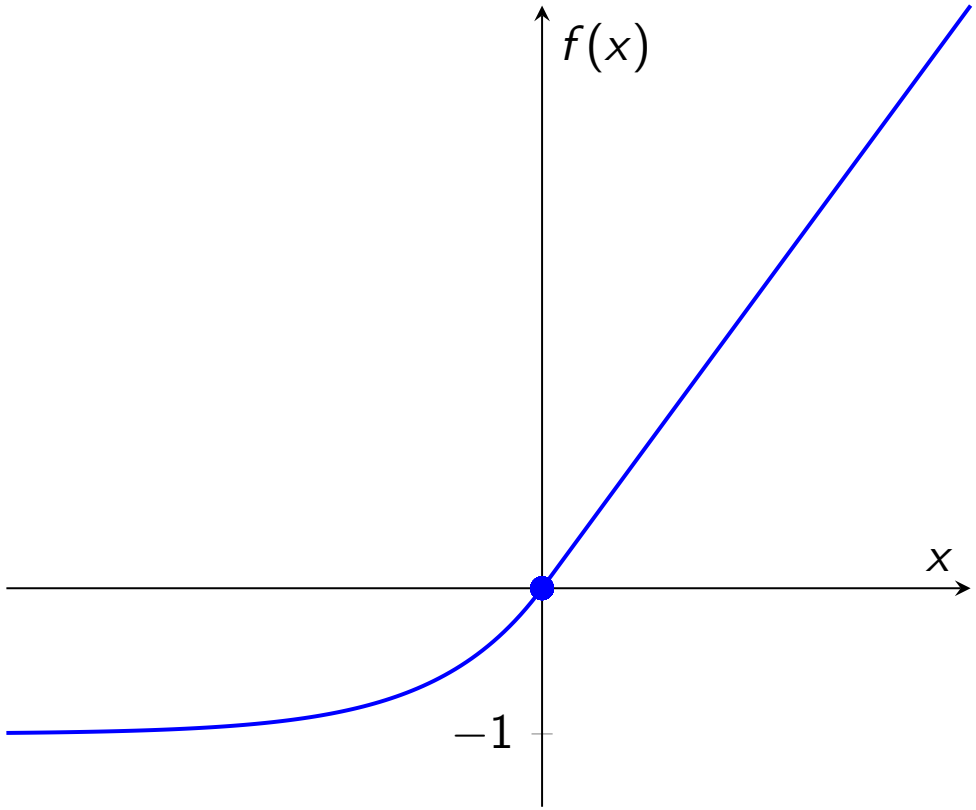
- Can be prone to **overfitting** on smaller datasets, as it adds more parameters to the model.
- Increases model complexity.

Use

- Used in more complex tasks where the extra learnable parameter might capture useful information.

8. Exponential Linear Unit (ELU) - Figure

ELU Function (example $\alpha = 1.0$)



$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$$

8. Exponential Linear Unit (ELU) - Analysis

Properties

- Smooth curve for negative inputs, unlike the sharp corner in ReLU.
- Becomes $-\alpha$ for large negative x.

Advantages

- **Solves "Dying ReLU" problem**.
- **Near zero-centered** output, which improves on standard ReLU.
- The smooth curve can help the optimizer converge faster.

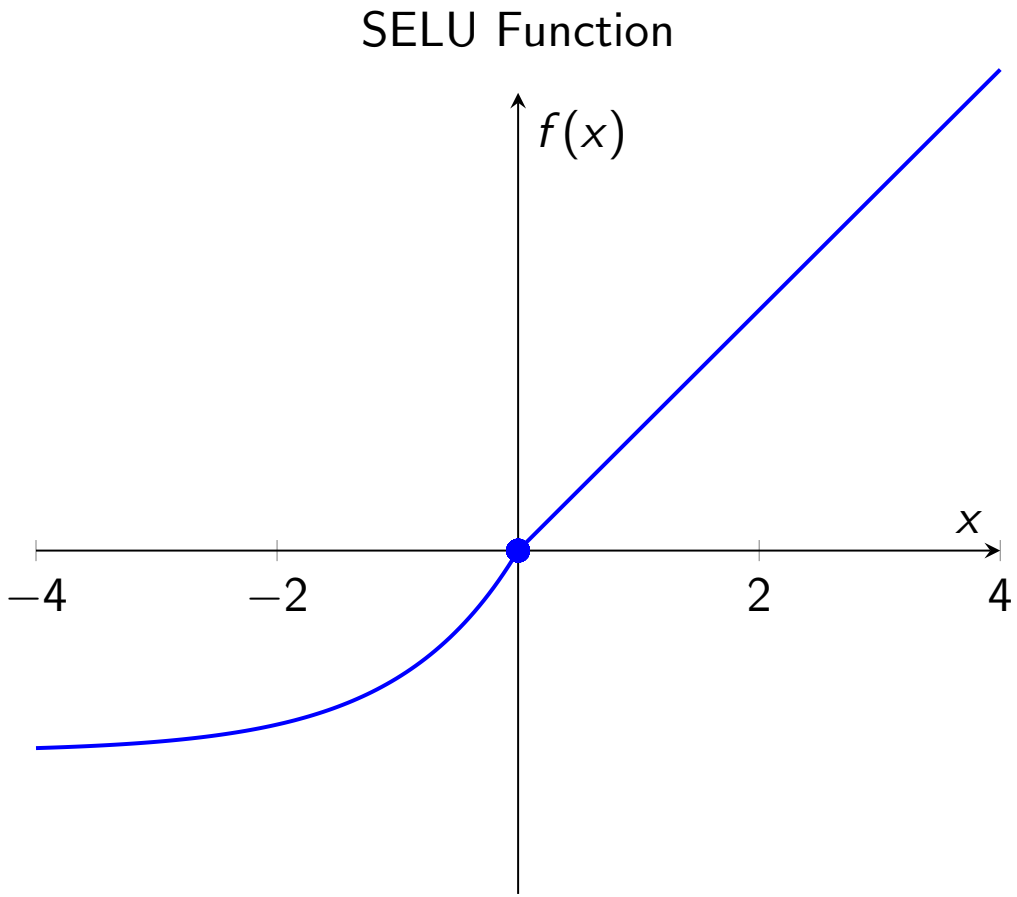
Drawbacks

- **More computationally expensive** than ReLU due to the exponential function (e^x).
- α is a hyperparameter to tune.

Use

- A strong alternative to ReLU, especially when faster convergence and better accuracy are needed (and compute cost is less of an issue).

9. Scaled ELU (SELU) - Figure



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

9. Scaled ELU (SELU) - Analysis

Properties

- A specific, scaled version of ELU.
- The values for λ and α are pre-defined constants ($\lambda \approx 1.0507$, $\alpha \approx 1.6732$).

Advantages

- **Self-Normalizing:** If weights are initialized correctly (Lecun Normal), SELU can push neuron activations towards a mean of 0 and std. dev. of 1.
- This "self-normalizing" property acts like internal batch normalization, **preventing vanishing/exploding gradients**.

Drawbacks

- Requires strict conditions: **Lecun Normal weight initialization** and **AlphaDropout** (not standard dropout).
- Not compatible with all network architectures (e.g., CNNs).

Use

- Designed for **deep Feed-Forward Networks (FNNs)**.

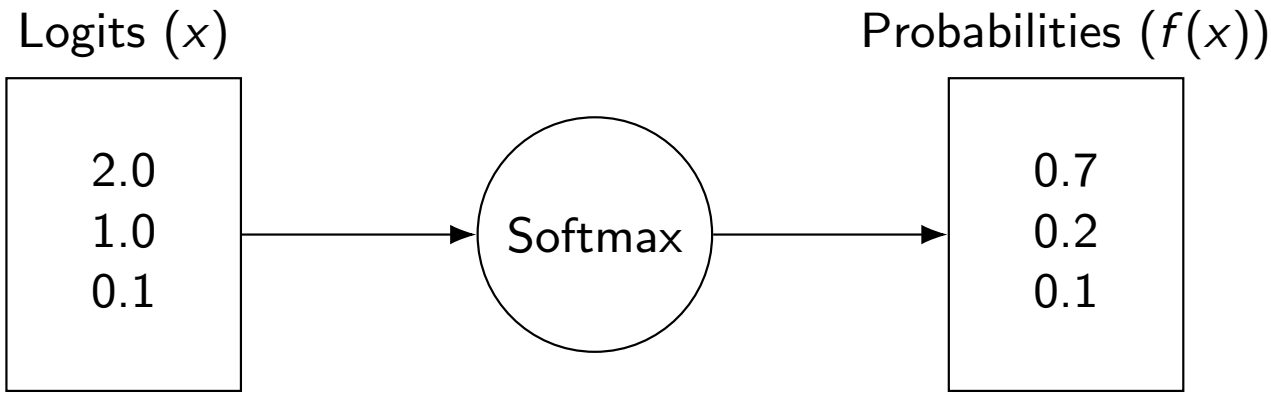
10. Softmax Function - Concept

Definition

This function is different: it's applied to a **vector** of inputs (the final layer's logits), not a single number. It converts a vector of raw scores into a **probability distribution**.

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

Where x_i is the i^{th} element of the input vector and K is the total number of classes.



Sum = 1.0

10. Softmax Function - Analysis

Properties & Advantages

- **Range:** Each output element is in (0, 1).
- **Sum to 1:** All elements in the output vector sum to 1.
- **Interpretable:** The output can be seen as the model's confidence or probability for each class.
- Differentiable.

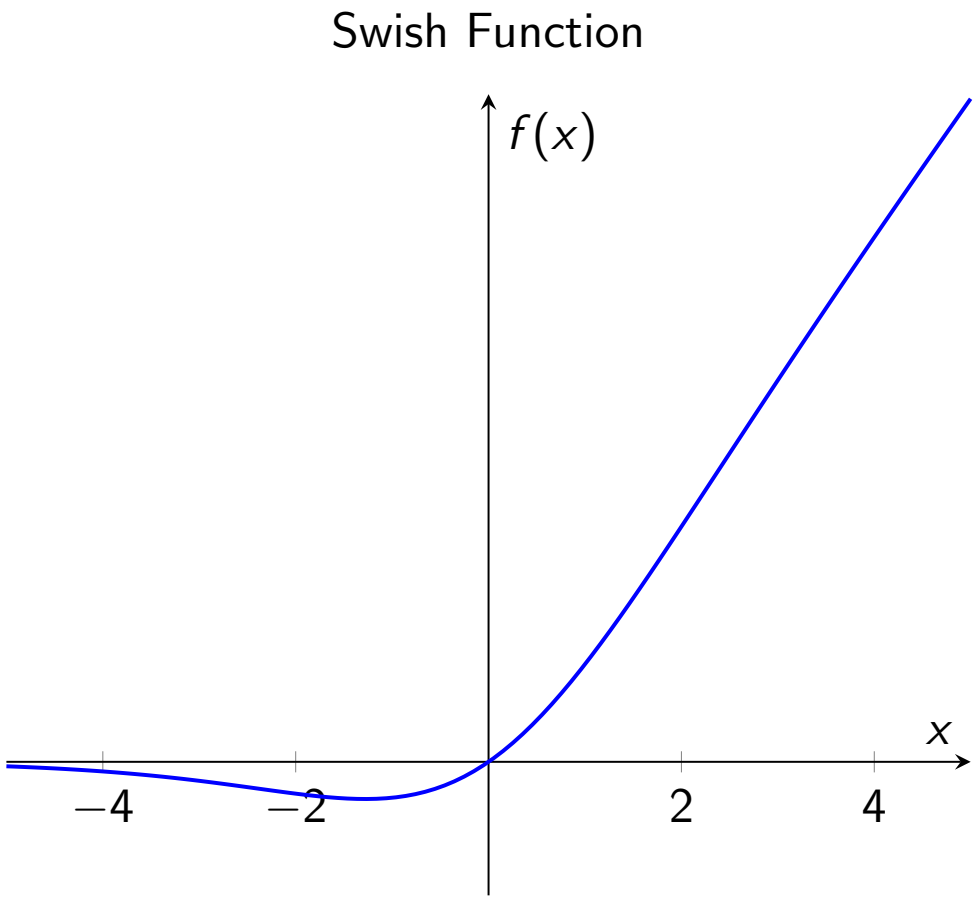
Drawbacks

- Can be computationally expensive (many exponentials).
- Can still suffer from saturation if input values (logits) are very large or very small.

Typical Use

- **Exclusively** used in the **output layer** for **multi-class classification** problems.

11. Swish - Figure



$$f(x) = x \cdot \sigma(x)$$

11. Swish - Analysis

Properties

- $f(x) = x \cdot \text{Sigmoid}(x)$.
- **Non-monotonic:** It dips slightly for negative values.
- Smooth and differentiable everywhere.
- Unbounded above, bounded below.

Advantages

- Tends to outperform ReLU on deeper networks.
- The non-monotonicity is claimed to help with learning.
- Avoids the "Dying ReLU" problem.

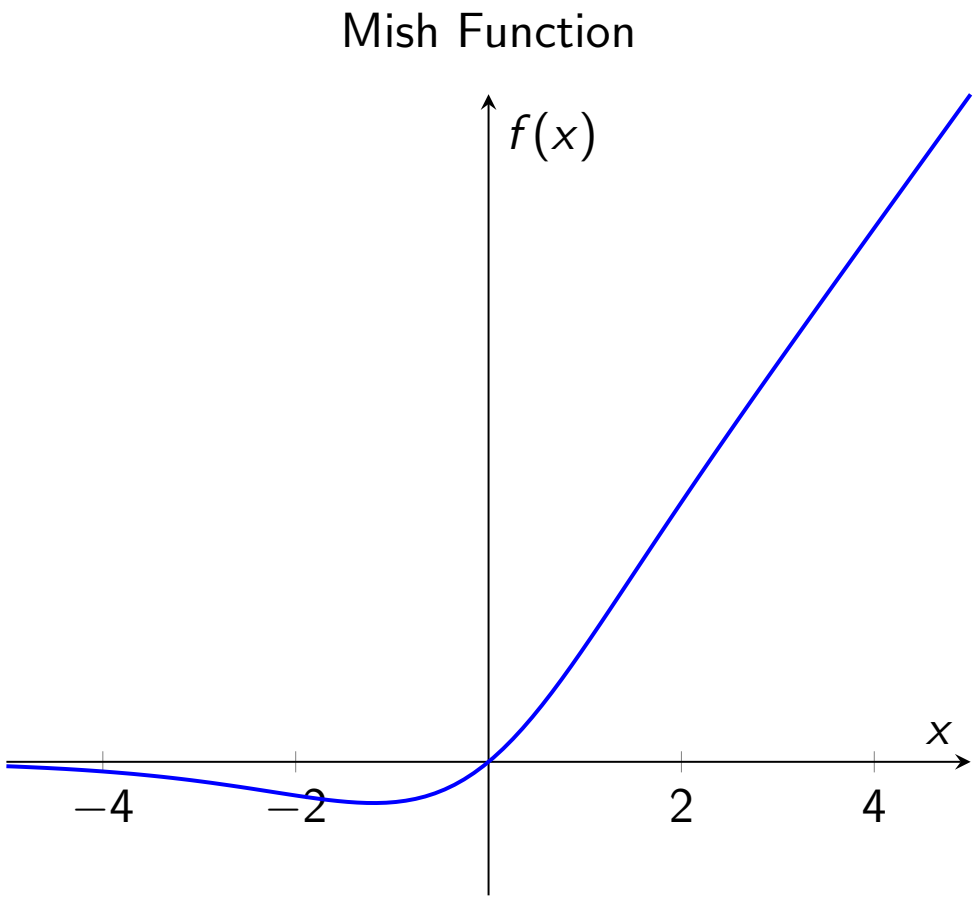
Drawbacks

- **Computationally expensive** (due to Sigmoid).

Use

- Developed by Google and used in their EfficientNet models ▸ Link.
- A strong, modern alternative to ReLU for deep networks.

12. Mish Function - Figure



$$f(x) = x \cdot \tanh(\text{softplus}(x))$$

12. Mish Function - Analysis

Properties

- softplus$(x) = \ln(1 + e^x)$.
- Similar shape to Swish (non-monotonic, smooth).
- Unbounded above, bounded below.

Advantages

- Reported to provide even better performance than Swish and ReLU in many deep learning tasks.
- Smoothness helps with optimization and generalization.

Drawbacks

- Even **more computationally expensive** than Swish due to the combination of tanh, ln, and e^x.

Use

- A state-of-the-art activation function used in object detection (e.g., YOLOv4) and other complex vision tasks.

How to Choose the Right Function?

General Rules of Thumb

- **Start with ReLU:** It's fast, efficient, and works well for most problems (especially CNNs). Be mindful of "Dying ReLU".
- **If ReLU fails:** Try **Leaky ReLU**, **ELU**, or **PReLU** to fix the "dying neuron" problem.
- **For deep FNNs:** Consider **SELU** (with Lecun initialization) to leverage self-normalization.
- **For RNNs: Tanh** or **Sigmoid** are still standard choices.
- **For state-of-the-art:** If compute is not a bottleneck, experiment with **Swish** or **Mish**.

How to Choose the Right Function?

General Rules of Thumb

- **Start with ReLU:** It's fast, efficient, and works well for most problems (especially CNNs). Be mindful of "Dying ReLU".
- **If ReLU fails:** Try **Leaky ReLU**, **ELU**, or **PReLU** to fix the "dying neuron" problem.
- **For deep FNNs:** Consider **SELU** (with Lecun initialization) to leverage self-normalization.
- **For RNNs: Tanh** or **Sigmoid** are still standard choices.
- **For state-of-the-art:** If compute is not a bottleneck, experiment with **Swish** or **Mish**.

For the Output Layer

The choice is dictated by the problem:

- **Binary Classification: Sigmoid**
- **Multi-Class Classification: Softmax**
- **Regression (any value): Linear**