Comprehensive Guide to Activation Functions in Neural Networks

Table of Contents

Introduction

Activation functions are mathematical functions that determine the output of neural network nodes. They introduce non-linearity into the network, enabling it to learn complex patterns and relationships in data. The choice of activation function significantly impacts the network's performance, training speed, and convergence.

Key Properties to Consider:

- **Zero-centered**: Whether the function's output is centered around zero
- **Monotonic**: Whether the function is strictly increasing

- **Differentiable**: Whether the function has a well-defined derivative
- **Range**: The output range of the function
- **Computational efficiency**: How fast the function can be computed

Linear Activation Function

Formula

$$f(x) = x$$

$$f'(x) = 1$$

Properties

- **Zero-centered**: Yes
- **Range**: (-∞, +∞)
- **Monotonic**: Yes
- **Differentiable**: Yes

Example Calculation

$$\text{Input: } x = -2, 0, 3, 5$$

$$\text{Output: } f(x) = -2, 0, 3, 5$$

$$\text{Derivative: } f'(x) = 1, 1, 1, 1$$

Usage

- **Recommended layers**: Output layer for regression problems
- **Not recommended for**: Hidden layers (no non-linearity)

Advantages

- Simple and fast computation
- No vanishing gradient problem
- Preserves input exactly

Disadvantages

- No non-linearity (network becomes linear regardless of depth)
- Cannot learn complex patterns
- Not suitable for hidden layers

Sigmoid Function

Formula

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

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

Properties

- **Zero-centered**: No (output range: 0-1)
- **Range**: (0, 1)
- **Monotonic**: Yes
- **Differentiable**: Yes

Example Calculation

$$\text{Input: } x = -2, 0, 2, 5$$

$$f(-2) = \frac{1}{1 + e^{2}} = \frac{1}{1 + 7.389} \approx 0.119$$

$$f(0) = \frac{1}{1 + e^{0}} = \frac{1}{1 + 1} = \frac{1}{2} = 0.5$$

$$f(2) = \frac{1}{1+e^{-2}} = \frac{1}{1+0.135} \approx 0.881$$

$$f(5) = \frac{1}{1+e^{-5}} = \frac{1}{1+0.007} \approx 0.993$$

Derivatives:

$$f'(-2) = 0.119 \times (1 - 0.119) \approx 0.105$$

$$f'(0) = 0.5 \times (1 - 0.5) = 0.25$$

$$f'(2) = 0.881 \times (1 - 0.881) \approx 0.105$$

$$f'(5) = 0.993 \times (1 - 0.993) \approx 0.007$$

Usage

- **Recommended layers**: Output layer for binary classification
- **Not recommended for**: Hidden layers in deep networks

Advantages

- Output bounded between 0 and 1
- Smooth gradient
- Good for probability interpretation

Disadvantages

- **Vanishing gradient problem**: Gradients become very small for large |x|
- **Not zero-centered**: Can cause zigzag dynamics during optimization
- **Computationally expensive**: Requires exponential calculation

Hyperbolic Tangent (Tanh)

Formula

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

$$f'(x) = 1 - \tanh^2(x) = 1 - [f(x)]^2$$

Properties

- **Zero-centered**: Yes
- **Range**: (-1, 1)
- **Monotonic**: Yes
- **Differentiable**: Yes

Example Calculation

$$\text{Input: } x = -2, 0, 2, 5$$

$$f(-2) = \tanh(-2) \approx -0.964$$

$$f(0) = \tanh(0) = 0$$

$$f(2) = \tanh(2) \approx 0.964$$

$$f(5) = \tanh(5) \approx 0.9999$$

Derivatives:

$$f'(-2) = 1 - (-0.964)^2 \approx 0.071$$

$$f'(0) = 1 - 0^2 = 1$$

$$f'(2) = 1 - (0.964)^2 \approx 0.071$$

$$f'(5) = 1 - (0.9999)^2 \approx 0.0002$$

Usage

- **Recommended layers**: Hidden layers in shallow networks, LSTM gates
- **Not recommended for**: Deep networks (vanishing gradient)

Advantages

- Zero-centered output
- Stronger gradients than sigmoid
- Good for shallow networks

Disadvantages

- **Vanishing gradient problem**: Similar to sigmoid but less severe
- **Computationally expensive**: Requires exponential calculations
- Still suffers from saturation

ReLU (Rectified Linear Unit)

Formula

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

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

Properties

- **Zero-centered**: No (output range: [0, +∞))

- **Range**: [0, +∞)
- **Monotonic**: Yes
- **Differentiable**: Almost everywhere (not at x=0)

Example Calculation

$$\text{Input: } x = -2, -0.5, 0, 2, 5$$

$$f(-2) = \max(0, -2) = 0$$

$$f(-0.5) = \max(0, -0.5) = 0$$

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

$$f(2) = \max(0, 2) = 2$$

$$f(5) = \max(0, 5) = 5$$

Derivatives:

$$f'(-2) = 0$$

$$f'(-0.5) = 0$$

$$f'(0) = 0 \text{ (by convention)}$$

$$f'(2) = 1$$

$$f'(5) = 1$$

Usage

- **Recommended layers**: Hidden layers in deep networks
- **Most popular choice**: Default activation for hidden layers

Advantages

- **Computationally efficient**: Simple max operation
- **No vanishing gradient**: For positive inputs
- **Sparse activation**: Many neurons output zero
- **Biological plausibility**: Similar to biological neurons

Disadvantages

- **Dying ReLU problem**: Neurons can become inactive forever
- **Not zero-centered**: Can cause optimization issues
- **Unbounded output**: Can lead to exploding gradients

Leaky ReLU

Formula

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

where α is typically 0.01

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

Properties

- **Zero-centered**: No
- **Range**: (-∞, +∞)
- **Monotonic**: Yes
- **Differentiable**: Almost everywhere

Example Calculation

$$\text{Input: } x = -2, -0.5, 0, 2, 5 \text{ (with } \alpha = 0.01)$$

$$f(-2) = \max(0.01 \times (-2), -2) = \max(-0.02, -2) = -0.02$$

$$f(-0.5) = \max(0.01 \times (-0.5), -0.5) = \max(-0.005, -0.5) = -0.005$$

$$f(0) = \max(0.01 \times 0, 0) = 0$$

$$f(2) = \max(0.01 \times 2, 2) = \max(0.02, 2) = 2$$

$$f(5) = \max(0.01 \times 5, 5) = \max(0.05, 5) = 5$$

Derivatives:

$$f'(-2) = 0.01$$

$$f'(-0.5) = 0.01$$

$$f'(0) = 0.01$$

$$f'(2) = 1$$

$$f'(5) = 1$$

Usage

- **Recommended layers**: Hidden layers when dying ReLU is a problem
- **Alternative to**: Standard ReLU

Advantages

- **Solves dying ReLU**: Small gradient for negative inputs
- **Computationally efficient**: Simple operation
- **No vanishing gradient**: For positive inputs

Disadvantages

- **Not zero-centered**: Similar to ReLU
- **Hyperparameter tuning**: Need to choose α
- **Inconsistent results**: Performance varies across tasks

Parametric ReLU (PReLU)

Formula

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

where α is learned during training

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

Properties

- **Zero-centered**: No
- **Range**: (-∞, +∞)
- **Monotonic**: Yes
- **Differentiable**: Almost everywhere

Example Calculation

Similar to Leaky ReLU, but α is learned (e.g., α might become 0.02, 0.1, etc.)

Usage

- **Recommended layers**: Hidden layers in deep networks
- **When to use**: When you want adaptive negative slopes

Advantages

- **Adaptive**: α is learned from data
- **Solves dying ReLU**: Like Leaky ReLU
- **Better performance**: Often outperforms Leaky ReLU

Disadvantages

- **Additional parameters**: Increases model complexity
- **Overfitting risk**: More parameters to learn
- **Computational overhead**: Slightly more expensive

ELU (Exponential Linear Unit)

Formula

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

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

Properties

- **Zero-centered**: Nearly (mean output close to zero)
- **Range**: (-α, +∞) where α > 0
- **Monotonic**: Yes
- **Differentiable**: Yes

Example Calculation

$$\text{Input: } x = -2, -0.5, 0, 2, 5 \text{ (with } \alpha = 1.0\text{)}$$

$$f(-2) = 1.0 \times (e^{-2} - 1) = 1.0 \times (0.135 - 1) = -0.865$$

$$f(-0.5) = 1.0 \times (e^{-0.5} - 1) = 1.0 \times (0.607 - 1) = -0.393$$

$$f(0) = 1.0 \times (e^{0} - 1) = 1.0 \times (1 - 1) = 0$$

$$f(2) = 2$$

$$f(5) = 5$$

Derivatives:

$$f'(-2) = 1.0 \times e^{-2} = 0.135$$

$$f'(-0.5) = 1.0 \times e^{-0.5} = 0.607$$

$$f'(0) = 1.0 \times e^{0} = 1.0$$

$$f'(2) = 1$$

$$f'(5) = 1$$

Usage

- **Recommended layers**: Hidden layers in deep networks
- **Alternative to**: ReLU when zero-centered output is desired

Advantages

- **Nearly zero-centered**: Better optimization dynamics

- **Smooth**: Differentiable everywhere
- **No dying neuron**: Always has non-zero gradient

Disadvantages

- **Computationally expensive**: Requires exponential calculation
- **Saturation**: Can saturate for very negative inputs
- **Hyperparameter**: Need to choose α

Swish

Formula

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

$$f'(x) = f(x) + \text{sigmoid}(x)(1 - f(x))$$

Properties

- **Zero-centered**: No
- **Range**: (-0.28, +∞) approximately
- **Monotonic**: No (has a small dip near x = -1.28)
- **Differentiable**: Yes

Example Calculation

$$\text{Input: } x = -2, 0, 2, 5$$

$$f(-2) = -2 \times \text{sigmoid}(-2) = -2 \times 0.119 = -0.238$$

$$f(0) = 0 \times \text{sigmoid}(0) = 0 \times 0.5 = 0$$

$$f(2) = 2 \times \text{sigmoid}(2) = 2 \times 0.881 = 1.762$$

$$f(5) = 5 \times \text{sigmoid}(5) = 5 \times 0.993 = 4.965$$

Usage

- **Recommended layers**: Hidden layers, especially in modern architectures
- **Popular in**: Google's research, mobile networks

Advantages

- **Self-gated**: Uses its own values to gate itself
- **Smooth**: Differentiable everywhere
- **Good performance**: Often outperforms ReLU

Disadvantages

- **Computationally expensive**: Requires sigmoid calculation
- **Not monotonic**: Can complicate optimization
- **Bounded below**: Has a lower bound

GELU (Gaussian Error Linear Unit)

Formula

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

where $\Phi(x)$ is the CDF of standard normal distribution

Approximation:

$$f(x) = \frac{1}{2}x\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$$

Properties

- **Zero-centered**: No

- **Range**: (-0.17, +∞) approximately
- **Monotonic**: No
- **Differentiable**: Yes

Example Calculation

Using approximation for $x = -1, 0, 1, 2$:

$$f(-1) \approx -0.159$$

$$f(0) = 0$$

$$f(1) \approx 0.841$$

$$f(2) \approx 1.954$$

Usage

- **Recommended layers**: Transformer models, BERT, GPT
- **Popular in**: Natural Language Processing

Advantages

- **Probabilistic**: Based on input's relationship to normal distribution
- **Smooth**: Better than ReLU variants
- **State-of-the-art**: Used in many successful models

Disadvantages

- **Computationally expensive**: Complex calculation
- **Not interpretable**: Less intuitive than other functions
- **Approximation needed**: Exact form is expensive

Softmax

Formula

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Properties

- **Zero-centered**: No
- **Range**: $(0, 1)$ and $\sum f(x_i) = 1$
- **Monotonic**: No
- **Differentiable**: Yes

Example Calculation

$$\text{Input vector: } \mathbf{x} = [2, 1, 3]$$

$$e^{\mathbf{x}} = [e^2, e^1, e^3] = [7.389, 2.718, 20.086]$$

$$\text{Sum} = 7.389 + 2.718 + 20.086 = 30.193$$

$$f(\mathbf{x}) = \left[\frac{7.389}{30.193}, \frac{2.718}{30.193}, \frac{20.086}{30.193}\right] = [0.245, 0.090, 0.665]$$

Usage

- **Recommended layers**: Output layer for multi-class classification
- **Essential for**: Probability distributions

Advantages

- **Probability interpretation**: Outputs sum to 1
- **Differentiable**: Good for gradient-based optimization
- **Multi-class**: Handles multiple classes naturally

Disadvantages

- **Computationally expensive**: Requires exponentials and normalization
- **Sensitive to outliers**: Large values dominate
- **Only for output**: Not suitable for hidden layers

Softplus

Formula

$$f(x) = \ln(1 + e^x)$$

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

Properties

- **Zero-centered**: No
- **Range**: (0, +∞)
- **Monotonic**: Yes
- **Differentiable**: Yes

Example Calculation

$$\text{Input: } x = -2, 0, 2, 5$$

$$f(-2) = \ln(1 + e^{-2}) = \ln(1 + 0.135) = \ln(1.135) \approx 0.127$$

$$f(0) = \ln(1 + e^0) = \ln(1 + 1) = \ln(2) \approx 0.693$$

$$f(2) = \ln(1 + e^2) = \ln(1 + 7.389) = \ln(8.389) \approx 2.127$$

$$f(5) = \ln(1 + e^5) = \ln(1 + 148.4) = \ln(149.4) \approx 5.007$$

Usage

- **Recommended layers**: Alternative to ReLU in some cases
- **Less common**: Not widely used

Advantages

- **Smooth**: Differentiable everywhere
- **Positive**: Always positive output
- **No dying neurons**: Always has gradient

Disadvantages

- **Computationally expensive**: Requires exponential and logarithm
- **Not zero-centered**: Similar issues to ReLU
- **Slower convergence**: Generally slower than ReLU

Mish

Formula

$$f(x) = x \cdot \tanh(\text{softplus}(x)) = x \cdot \tanh(\ln(1 + e^x))$$

Properties

- **Zero-centered**: No
- **Range**: (-0.31, +∞) approximately
- **Monotonic**: No
- **Differentiable**: Yes

Example Calculation

Input: $x = -1, 0, 1, 2$

For $x = 1$:
$\text{softplus}(1) = \ln(1 + e^1) = \ln(1 + 2.718) = \ln(3.718) \approx 1.313$

$\tanh(1.313) \approx 0.865$
$f(1) = 1 \times 0.865 = 0.865$

For $x = 0$:
$f(0) = 0 \times \tanh(\ln(1 + e^0)) = 0 \times \tanh(\ln(2)) = 0$

For $x = -1$:
$\text{softplus}(-1) = \ln(1 + e^{-1}) = \ln(1 + 0.368) = \ln(1.368) \approx 0.313$
$\tanh(0.313) \approx 0.303$
$f(-1) = -1 \times 0.303 = -0.303$

Derivatives:
For $x = 1$:
$f'(1) = \tanh(1.313) + 1 \times \text{sech}^2(1.313) \times \text{sigmoid}(1)$
$= 0.865 + 1 \times 0.252 \times 0.731 = 0.865 + 0.184 = 1.049$

Usage

- **Recommended layers**: Hidden layers in modern networks
- **Recent research**: Gaining popularity

Advantages

- **Self-regularizing**: Has regularization properties
- **Smooth**: Better than ReLU variants
- **Good performance**: Often outperforms Swish and ReLU

Disadvantages

- **Computationally expensive**: Most complex among common functions
- **Memory intensive**: Requires storing intermediate values
- **New**: Less tested than established functions

Comparison Table

Function	Zero-Centered	Range	Computational Cost	Vanishing Gradient	Common Issues
Linear	✅	(-∞, +∞)	Very Low	No	No non-linearity
Sigmoid	❌	(0, 1)	High	✅	Vanishing gradient, not zero-centered
Tanh	✅	(-1, 1)	High	✅	Vanishing gradient
ReLU	❌	[0, +∞)	Very Low	No	Dying neurons, not zero-centered
Leaky ReLU	❌	(-∞, +∞)	Very Low	No	Not zero-centered
PReLU	❌	(-∞, +∞)	Low	No	Additional parameters
ELU	Nearly	(-α, +∞)	Medium	No	Computational cost
Swish	❌	(-0.28, +∞)	High	No	Not monotonic
GELU	❌	(-0.17, +∞)	High	No	Complex computation
Softmax	❌	(0, 1), Σ=1	High	Potential	Only for output
Softplus	❌	(0, +∞)	High	No	Computational cost
Mish	❌	(-0.31, +∞)	Very High	No	Very expensive

Layer-wise Recommendations

Hidden Layers

Deep Networks (>5 layers):

- **Primary choice**: ReLU
- **If dying ReLU occurs**: Leaky ReLU, ELU, or PReLU
- **For better performance**: Swish, GELU, or Mish
- **Avoid**: Sigmoid, Tanh (vanishing gradient)

Shallow Networks (≤5 layers):

- **Good choices**: Tanh, ReLU, ELU
- **For zero-centered**: Tanh, ELU
- **For simplicity**: ReLU

Convolutional Layers:

- **Standard**: ReLU
- **Advanced**: Swish, Mish
- **Mobile/Efficient**: ReLU, Leaky ReLU

Recurrent Networks:

- **LSTM/GRU gates**: Sigmoid, Tanh
- **Hidden states**: Tanh, ReLU

Output Layers

Binary Classification:

- **Standard**: Sigmoid
- **Alternative**: Tanh (with appropriate interpretation)

Multi-class Classification:

- **Standard**: Softmax
- **Required**: For probability interpretation

Regression:

- **Unbounded**: Linear, ReLU
- **Bounded**: Sigmoid, Tanh
- **Positive values**: ReLU, Softplus

Multi-label Classification:

- **Standard**: Sigmoid (applied to each output)

Common Problems and Solutions

Problem 1: Vanishing Gradient

Symptoms: Training becomes very slow, early layers don't learn
Affected functions: Sigmoid, Tanh
Solutions:

- Use ReLU, Leaky ReLU, or ELU
- Apply batch normalization
- Use residual connections
- Reduce network depth

Problem 2: Dying ReLU

Symptoms: Many neurons output zero, gradients become zero
Affected functions: ReLU
Solutions:

- Use Leaky ReLU ($\alpha = 0.01$)
- Use ELU or PReLU
- Reduce learning rate
- Better weight initialization
- Use batch normalization

Problem 3: Exploding Gradient

Symptoms: Loss increases rapidly, weights become very large
Affected functions: Any unbounded function (ReLU variants)

Solutions:

- Gradient clipping
- Lower learning rate
- Better weight initialization
- Use bounded functions (Sigmoid, Tanh)

Problem 4: Slow Convergence

Symptoms: Training takes very long to converge
Affected functions: Sigmoid, Tanh, Softplus
Solutions:

- Use ReLU or its variants
- Apply batch normalization
- Use adaptive learning rates (Adam, RMSprop)
- Better weight initialization

Problem 5: Not Zero-Centered

Symptoms: Zigzag optimization patterns, slower convergence
Affected functions: ReLU, Sigmoid, Swish
Solutions:

- Use Tanh or ELU
- Apply batch normalization
- Use zero-centered initialization

Best Practices

1. Default Choices

- **Hidden layers**: Start with ReLU
- **Output layer**: Softmax (classification), Linear (regression)
- **If ReLU fails**: Try Leaky ReLU or ELU

2. Experimentation Order

1. ReLU → Leaky ReLU → ELU
2. If performance critical: Swish → GELU → Mish
3. For specific domains: Research domain-specific functions

3. Considerations by Network Type

- **CNNs**: ReLU, Swish
- **RNNs**: Tanh (hidden), Sigmoid (gates)
- **Transformers**: GELU, Swish
- **GANs**: Leaky ReLU, Tanh

4. Hyperparameter Tuning

- **Leaky ReLU**: α ∈ [0.01, 0.3]
- **ELU**: α ∈ [0.1, 2.0]
- **Always validate**: Use validation set to compare

5. Implementation Tips

- Use vectorized operations
- Consider memory usage for complex functions
- Profile computational overhead
- Use mixed precision when possible

6. Mathematical Considerations

Gradient Flow Analysis:
For a function f with derivative f', the gradient update is:
$\Delta w = -\eta \cdot f'(z) \cdot \frac{\partial L}{\partial a}$

Where:

- η is the learning rate
- z is the pre-activation
- $a = f(z)$ is the activation
- L is the loss function

Key Insights:

- Functions with $f'(z) \approx 0$ cause vanishing gradients
- Functions with large $f'(z)$ can cause exploding gradients
- Zero-centered functions help maintain gradient magnitudes

7. Initialization Compatibility

Different activation functions work best with specific weight initialization schemes:

Xavier/Glorot Initialization:
$w \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{in}+n_{out}}}\right)$

- **Best for**: Tanh, Sigmoid, Linear

He Initialization:
$w \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{in}}}\right)$

- **Best for**: ReLU, Leaky ReLU, ELU

LeCun Initialization:
$w \sim \mathcal{N}\left(0, \sqrt{\frac{1}{n_{in}}}\right)$

- **Best for**: SELU (Self-Normalizing Neural Networks)

8. Activation Function Selection Flowchart

```
Start
  ↓
Is it output layer?
  ↓ Yes → Binary Classification? → Yes → Sigmoid
  |         ↓ No
  |       Multi-class Classification? → Yes → Softmax
  |         ↓ No
  |       Regression → Linear/ReLU
  ↓ No
Hidden Layer
  ↓
Deep Network (>5 layers)?
  ↓ Yes → Try ReLU → Dying ReLU problem? → Yes → Leaky ReLU/ELU
  |         ↓ No                            ↓ No
  |       Performance critical? → Yes → Swish/GELU/Mish
  |         ↓ No                   ↓ No
  |       Keep ReLU             Keep ReLU
  ↓ No
Shallow Network → Tanh/ReLU → Zero-centered needed? → Yes → Tanh/ELU
                    ↓ No                                ↓ No
                  ReLU                                ReLU
```

9. Performance Benchmarking

When comparing activation functions, measure:

Training Metrics:

- Convergence speed (epochs to target accuracy)
- Training stability (loss variance)
- Gradient magnitudes over time

Validation Metrics:

- Final accuracy/loss
- Generalization gap
- Robustness to hyperparameters

Computational Metrics:

- Forward pass time per batch
- Backward pass time per batch
- Memory usage

10. Domain-Specific Recommendations

Computer Vision:

- **CNNs**: ReLU (standard), Swish (advanced)
- **Object Detection**: ReLU, Leaky ReLU
- **Generative Models**: Leaky ReLU, Tanh

Natural Language Processing:

- **Transformers**: GELU (BERT, GPT), Swish
- **RNNs**: Tanh (hidden states), Sigmoid (gates)
- **Language Models**: GELU, ReLU

Time Series Analysis:

- **LSTMs**: Sigmoid (gates), Tanh (cell state)
- **GRUs**: Sigmoid (gates), Tanh (candidate)
- **Feed-forward**: ReLU, ELU

Reinforcement Learning:

- **Policy Networks**: Tanh (continuous), Softmax (discrete)
- **Value Networks**: ReLU, Leaky ReLU
- **Actor-Critic**: Mixed (Tanh for actor, ReLU for critic)

This comprehensive guide should help you choose the right activation function for your neural network architecture and specific use case. Remember to always validate your choices empirically, as the best activation function can vary depending on your specific dataset and problem domain.