1. Activation Function

An activation function is a key components in neural networks that decides the output of a neuron based on its inputs. It adds non-linearity to the networks, allowing it to model complex relationships in data. Common activation functions include:

Comparison of Activation Functions

Activation Function	Formula	Output Range	Key Characteristics
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$(0,1)$	• Converts input to probabilities • Smooth curve • Prone to vanishing gradients for large inputs
ReLu	$f(x) = \max(0, x)$	$[0, \infty)$	• Computationally efficient • Avoids vanishing gradient for positive inputs • Prone to dying ReLu
Tanh	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$(-1,1)$	• Outputs are zero-centered • Better than sigmoid in hidden layers • Suffers from vanishing gradients
Leakly ReLu	$f(x) = x\ if\ x > 0, 0.01x\ otherwise$	$(-\infty, \infty)$	• Mitigates dying ReLu by allowing small gradients for negative inputs • Requires slope tuning

Use Cases and Limitation

1. **Sigmoid:**
   - The **sigmoid function** is commonly used for binary classification as it outputs values between 0 and 1, representing probabilities. Its smooth curve makes it useful for logistic regression and as an activation function in shallow networks.

   Limitation:

   - Causes vanishing gradient problem, making it unsuitable for deep networks

2. **ReLu:**
   - Widely used in hidden layers of deep learning models, especially in CNNs and feedforward networks.

   Limitation:

   - Suffers from the dying ReLu problem, where some neurons stop updating during training.

3. **Tanh:**
   - Used in RNNs and other architectures where zero-centered outputs are beneficial.

   Limitation:

   - Computationally expensive and inefficient for sparse representations compared to ReLU.

4. **Leaky ReLu**
   - Suitable for deep networks where the dying ReLu Problem is observed

   Limitations:

   - Less commonly used than standard ReLu in practice

2. Optimization Algorithm

An **optimization algorithm** is a method used to adjust a neural network's weights and biases to minimize the loss function during training. It ensures efficient convergence by determining the direction and size of weight updates. Under is a detailed comparison of three widely used algorithms:

Comparison of Optimization Algorithm

Algorithm	**Description**	**Pros**	**Cons**
SGD	Updates weights using the gradient of a mini-batch	Simple, memory-efficient	Sensitive to learning rate; slow convergence
Adam	Combines momentum and adaptive learning rates(uses moving averages of gradient and its square)	Fast convergence; adaptive learning rates	Require more memory; many hyperparameters
RMSprop	Uses moving average of squared gradients to scale learning rate	Works well for RNNs and non-stationary problems	Can converge too quickly to suboptimal values

Impact of Learning Rate

The impact of learning rate determines the magnitude of weight adjustments during training; a high rate might result in overshooting the optimal values, while a low rate could make the training process slower to converge. Proper tuning is crucial for balancing speed and stability in optimization.

- High Learning Rate: Rapid updates but risk overshooting the minimum, leading to divergence
- Low Learning Rate: Safer convergence but requires more epocs and may get stuck in local minima

Addressing Learning Rate Issue

1. **Learning Rate Scheduling**
    - Gradually reduces the learning rate during training(e.g.., exponential decay, step decay)
2. **Adaptive Methods**
    - The Adam and RMSprop adjust learning rate dynamically based on gradient magnitudes, allowing for efficient learning even with poorly chosen initial values
3. **Warm Restarts**
    - The warm restarts periodically remit the learning rate to help escape local minima and explore the loss landscape