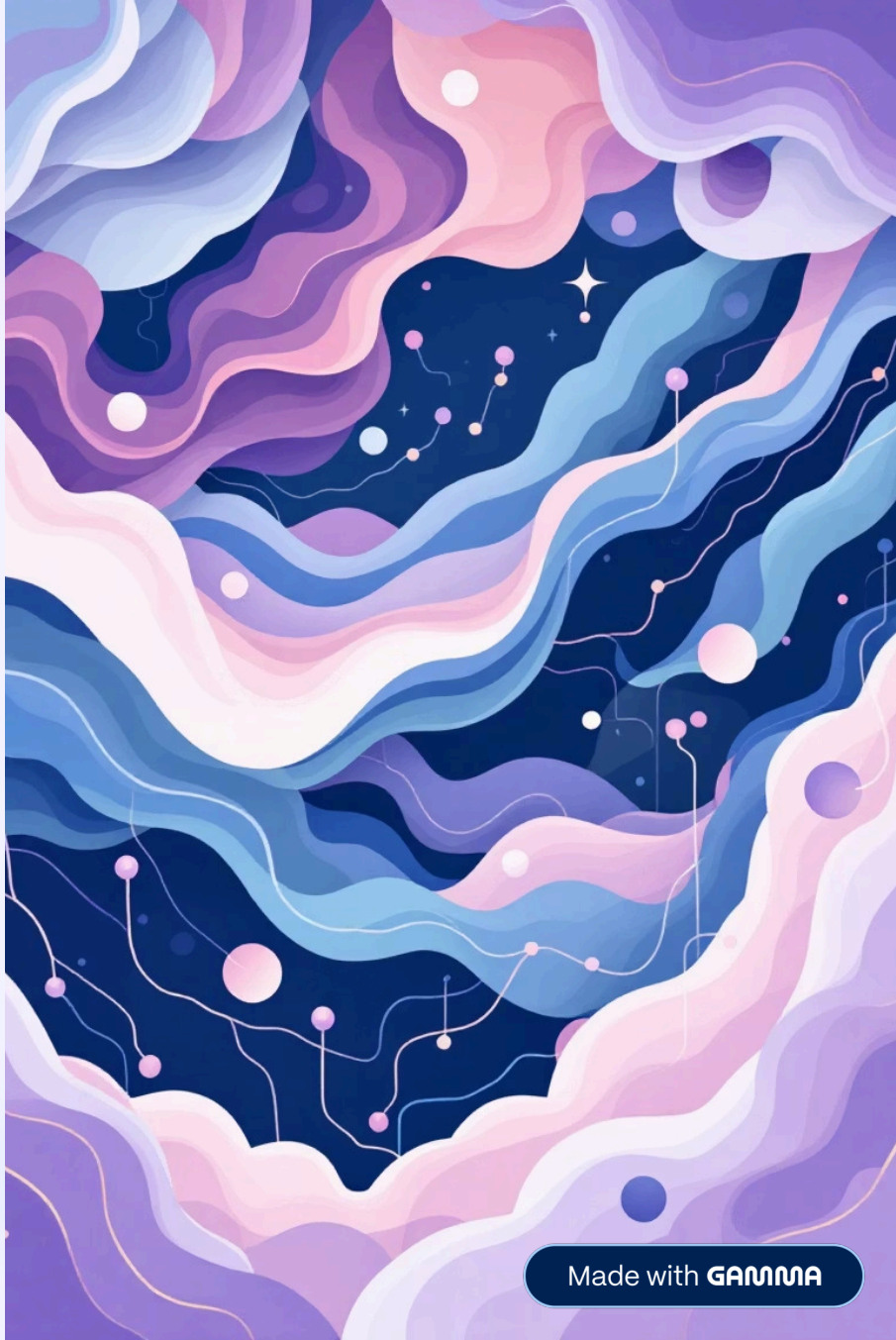
Step 4 – Add & Norm

Making Transformer Training Stable and Efficient



Why "Add & Norm"?

Deep networks like Transformers present significant training challenges that can severely impact model performance and convergence. Understanding these challenges is crucial for appreciating why the Add & Norm mechanism is so fundamental to modern Transformer architectures.

The primary issues that plague deep neural network training include vanishing gradients, where gradient signals become progressively weaker as they propagate backwards through layers, and unstable learning dynamics, where parameter updates fluctuate dramatically between training iterations. These problems become exponentially worse as network depth increases, making very deep architectures nearly impossible to train effectively without proper stabilisation techniques.

Vanishing Gradients

As gradients propagate backwards through multiple layers, they multiply repeatedly, becoming infinitesimally small. This causes early layers to learn extremely slowly or not at all.

Unstable Learning

Parameter values can swing wildly between updates, preventing convergence to optimal solutions and making training unpredictable and difficult to tune.

The Solution: By combining **Residual Connections** with **Layer Normalisation**, we create the powerful **Add & Norm** mechanism that addresses both challenges simultaneously, enabling the training of very deep Transformer models.

What Happens in Add & Norm

The Add & Norm operation is conceptually elegant yet remarkably effective. It operates on two key inputs at each sub-layer within the Transformer architecture, combining them in a specific mathematical way that preserves information whilst stabilising training dynamics.

01

Original Input (x)

The input tensor entering the sub-layer, containing all information from previous processing stages. This represents the "residual path" that will bypass the transformation.

02

Sub-Layer Transformation

The output from the current sub-layer operation, such as Multi-Head Attention or Feed-Forward Network, denoted as SubLayer(x). This is the "main path" where actual feature learning occurs.

03

Addition Operation

The original input x is added element-wise to the sub-layer output, creating z = x + SubLayer(x). This preserves the identity mapping whilst allowing learned transformations.

04

Layer Normalisation

The combined result undergoes normalisation to stabilise activations, producing the final output: LayerNorm(x + SubLayer(x)). This ensures consistent scale across layers.

Mathematical Formulation: Output = LayerNorm(x + SubLayer(x))

✅ "Add" → adds back original input for gradient flow
✅ "Norm" → normalises result for training stability

Step 1 – Residual (Skip) Connection

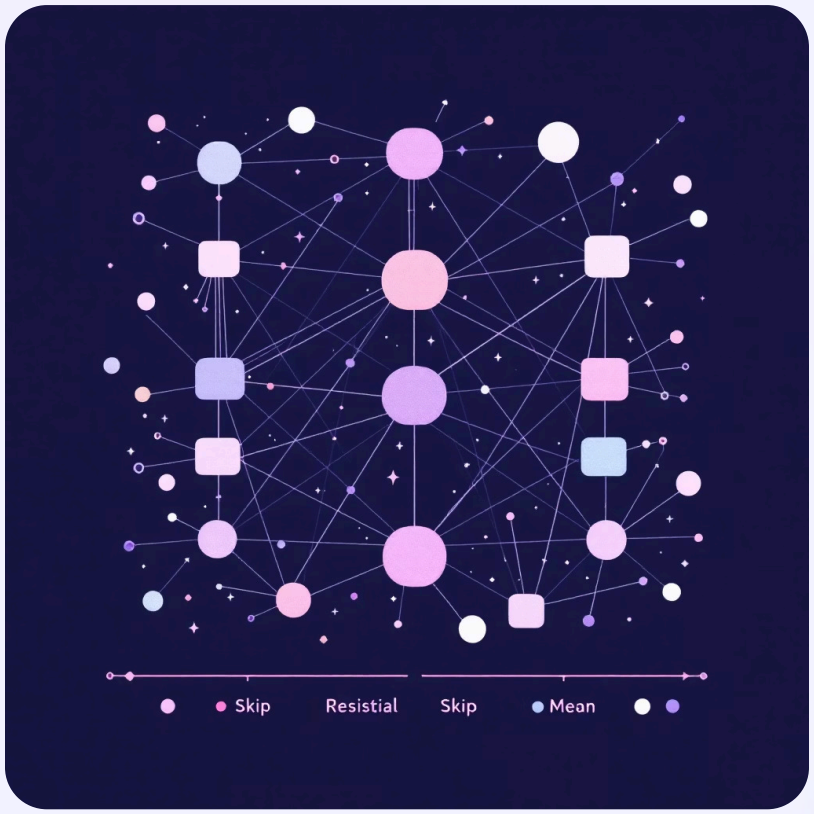
The Core Concept

Residual connections, also known as skip connections, represent one of the most influential innovations in deep learning architecture design. The fundamental idea is deceptively simple: rather than forcing each layer to learn the desired underlying mapping directly, we allow it to learn the *residual* or difference from the identity mapping.

Mathematically, for an attention sub-layer, this is expressed as:

$$z = x + \text{Attention}(x)$$

This seemingly minor modification has profound implications for training dynamics and model capability.



Why Residual Connections Work

The effectiveness of residual connections stems from several complementary mechanisms that work together to enable training of very deep networks:

- **Information Preservation:** The original input x passes unchanged through the addition, ensuring that information from earlier layers is never completely lost or overwritten by subsequent transformations.
- **Gradient Highway:** During backpropagation, gradients can flow directly through the skip connection without passing through non-linear activations, preventing the vanishing gradient problem.
- **Feature Retention:** Earlier learned representations remain accessible to deeper layers, allowing the network to build upon previous learning rather than starting afresh at each layer.
- **Faster Convergence:** The identity path provides a strong baseline, allowing the network to learn refinements rather than complete transformations, which accelerates training significantly.

Historical Context

Origin: Residual connections were popularised by **ResNet** (Residual Networks) in 2015, which revolutionised computer vision by enabling networks with 152+ layers.

The Transformer architecture adopted this proven technique, demonstrating its generality across different domains and architectures.

Step 2 – Layer Normalisation

After adding the input to the sub-layer output through the residual connection, we apply layer normalisation to the combined result. This second critical component of the Add & Norm mechanism ensures that activations remain in a stable range throughout training, regardless of network depth or the magnitude of parameter updates.

The Normalisation Process

Layer normalisation operates on the combined tensor z that results from the addition step. The operation can be expressed in two equivalent ways:

$$\text{Output} = \text{LayerNorm}(z)$$

Or, showing the full composition:

$$\text{Output} = \text{LayerNorm}(x + \text{Attention}(x))$$

The normalisation process itself involves several steps: computing the mean and variance across the feature dimension for each individual sample, subtracting the mean and dividing by the standard deviation to achieve zero mean and unit variance, and finally applying learnable scaling (γ) and shifting (β) parameters to restore representational capacity.

Key Properties

Zero-Centred

Ensures activations have mean = 0, preventing systematic bias

Unit Variance

Maintains variance = 1, keeping activations in a consistent range

Training Acceleration

Reduces internal covariate shift, allowing higher learning rates

The combination of residual connections and layer normalisation creates a powerful synergy: residual connections provide gradient highways and information preservation, whilst layer normalisation ensures that the activations flowing through these highways remain well-behaved and stable throughout the training process.

Why Normalise?

Normalisation is not merely a numerical convenience—it fundamentally addresses the challenge of **internal covariate shift**, where the distribution of layer inputs changes during training as parameters in previous layers are updated. This shifting distribution makes learning difficult because each layer must constantly adapt to a moving target.

10

Mean = 0 (Zero-Centred)

By centring activations around zero, we prevent systematic bias in either the positive or negative direction. This symmetry helps optimisers like Adam and SGD converge more reliably, as they don't need to compensate for consistent directional drift in the activations. Zero-centring also ensures that the non-linear activation functions (like ReLU) operate in their most effective range.

1

Variance = 1 (Unit Variance)

Maintaining unit variance ensures that activation magnitudes remain consistent across layers and throughout training. Without this, activations could explode (growing exponentially with depth) or vanish (shrinking to near-zero), both of which make training extremely difficult. Unit variance provides a stable foundation that allows gradients to flow effectively.

Faster Training

With normalised activations, the loss landscape becomes smoother and less prone to sharp valleys or saddle points. This allows the use of higher learning rates without instability, dramatically accelerating convergence. Models can achieve the same performance in significantly fewer training iterations.

Stable Learning

Normalisation acts as a regulariser, reducing the model's sensitivity to parameter initialisation and making training more robust to hyperparameter choices. This stability means training runs are more reproducible and less likely to diverge or get stuck in poor local minima, making the entire training process more reliable.

LayerNorm vs BatchNorm

Whilst both Layer Normalisation and Batch Normalisation aim to stabilise training through normalisation, they differ fundamentally in *which dimensions* they normalise across. Understanding these differences is crucial for appreciating why LayerNorm is the preferred choice for Transformer architectures.

Feature	Batch Normalisation	Layer Normalisation
Normalises Across	Batch dimension (across samples in a mini-batch)	Feature dimension (across features within a single sample)
Depends on Batch Size?	Yes – requires sufficient batch size for reliable statistics	No – computes statistics per sample independently
Training vs Inference	Different behaviour: uses moving averages during inference	Identical behaviour: same computation always
Commonly Used In	CNNs for computer vision, some RNN applications	Transformers, NLP models, sequence processing
Variable Sequence Lengths?	❌ Problematic with varying lengths	✅ Handles variable lengths naturally
Performance Impact	Can be slower due to batch statistics computation	Generally faster, especially with small batches

---

Why Transformers Use LayerNorm

Sequence Length Variability: NLP tasks involve sequences of varying lengths, making batch-wise normalisation problematic. LayerNorm handles this naturally by computing statistics per sample.

Batch Size Independence: During inference or with small batch sizes, LayerNorm maintains consistent behaviour, whilst BatchNorm's performance can degrade significantly.

Computational Efficiency: For typical Transformer workloads with large feature dimensions, LayerNorm is more efficient and easier to implement across distributed training setups.

Symbol Explanation

Understanding the mathematical notation used in the Add & Norm formulation is essential for implementing and debugging Transformer models. Here's a comprehensive breakdown of each symbol and its role in the architecture.

x (Input Vector)

Represents: The input tensor to the sub-layer, containing the representation from the previous layer.

Shape: Typically [batch_size, sequence_length, d_model] where d_model is the embedding dimension (512 or 768 commonly).

Role: Serves as both the input to the transformation and the residual connection that bypasses it.

SubLayer(x)

Represents: The output from a sub-layer transformation, which could be Multi-Head Attention or Feed-Forward Network.

Operation: A learnable transformation that processes the input and produces new representations.

Shape: Same as input x, maintaining [batch_size, sequence_length, d_model] for residual addition.

z (Intermediate Sum)

Represents: The element-wise sum of the input x and the sub-layer output SubLayer(x).

Formula: $z = x + \text{SubLayer}(x)$

Purpose: Combines the residual path with the transformation path before normalisation.

LayerNorm

Represents: The layer normalisation operation that standardises activations.

Process: Computes mean and variance across features, normalises to mean=0 and variance=1, then applies learnable scale (γ) and shift (β) parameters.

Implementation: Applied independently to each position in the sequence.

Output (Final Result)

Represents: The normalised, stable output ready for the next layer.

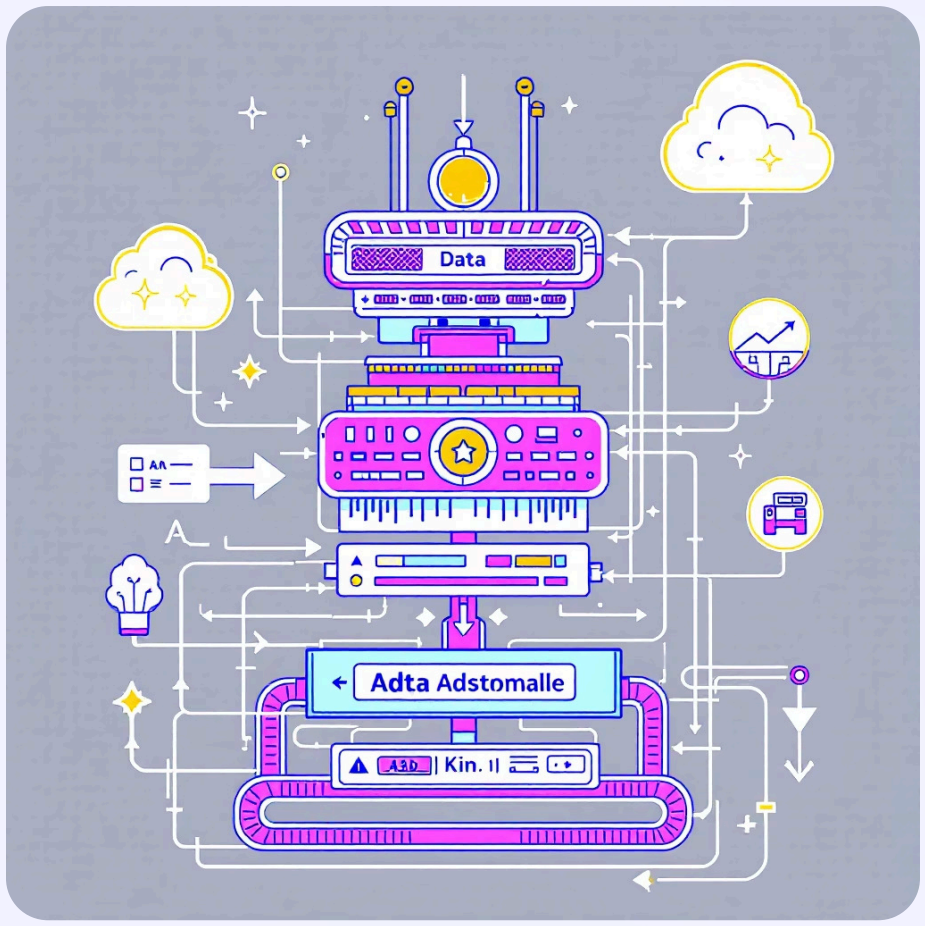
Formula: $\text{Output} = \text{LayerNorm}(x + \text{SubLayer}(x))$

Properties: Maintains the same shape as input, with stabilised activations suitable for deep network training.

Implementation Note: In practice, LayerNorm includes learnable parameters γ (scale) and β (shift) for each feature dimension, allowing the model to adjust the normalised distribution when beneficial for the task.

Where Add & Norm Appears

The Add & Norm mechanism is not applied just once, but **twice** within each Transformer encoder or decoder block. This repeated application ensures that both major transformations—attention and feed-forward processing—benefit from improved gradient flow and training stability.



The Two Applications

After Multi-Head Attention

The first Add & Norm follows the attention mechanism, stabilising the learned attention patterns and preserving the input representation.

$$\text{Output}_1 = \text{LayerNorm}(x + \text{MultiHeadAttention}(x))$$

After Feed-Forward Network

The second Add & Norm follows the position-wise feed-forward network, ensuring stable non-linear transformations.

$$\text{Output}_2 = \text{LayerNorm}(\text{Output}_1 + \text{FFN}(\text{Output}_1))$$

Complete Flow Through a Transformer Block

1. **Input Embedding**
   Token embeddings combined with positional encodings enter the block
2. **Multi-Head Attention**
   Computes attention across sequence positions, learning contextual relationships
3. **Add & Norm (First)**
   Residual connection + normalisation stabilises attention output
4. **Feed-Forward Network**
   Two-layer MLP with ReLU activation processes each position independently
5. **Add & Norm (Second)**
   Residual connection + normalisation stabilises FFN output
6. **Output to Next Block**
   Stabilised representations pass to the next encoder/decoder layer

This pattern repeats N times (typically 6-12) in both the encoder and decoder stacks, creating very deep networks that remain trainable thanks to the Add & Norm mechanism at each sub-layer.

Quick Recap

The Add & Norm mechanism is a cornerstone of Transformer architecture design, elegantly solving the challenges of training very deep neural networks through two complementary operations.

1️⃣ Add (Residual Connection)

What it does: Adds the original input back to the sub-layer output through element-wise addition

Key benefit: Creates gradient highways that prevent vanishing gradients and preserve information from earlier layers

Formula: z = x + SubLayer(x)

2️⃣ Norm (Layer Normalisation)

What it does: Normalises the combined result to have mean=0 and variance=1 across feature dimensions

Key benefit: Stabilises training by reducing internal covariate shift and enabling higher learning rates

Formula: Output = LayerNorm(z)

Overall Benefits

- **Superior Gradient Flow:** Residual connections provide direct paths for gradients to flow through the network
- **Faster Convergence:** Normalisation smooths the loss landscape, allowing faster, more stable optimisation
- **Deeper Networks:** Enables training of very deep Transformers (12+ layers) that would otherwise be impossible
- **Better Generalisation:** The regularisation effect of normalisation improves model performance on unseen data

Usage Pattern

Applied Twice Per Block

Once after Multi-Head Attention, once after Feed-Forward Network

Repeated Across Layers

Each of the N encoder/decoder layers uses Add & Norm twice

Critical for Deep Models

Becomes increasingly important as network depth increases

Key Takeaway: The synergy between residual connections and layer normalisation is what makes modern Transformers trainable and effective. Together, they address the fundamental challenges of deep learning—vanishing gradients and training instability—enabling the powerful language models we use today.