Transformer Architecture & Self-Attention

Deep Dive into 'Attention is All You Need'

1. The Transformer Model

The Transformer, introduced by Vaswani et al. in 2017 in the landmark paper 'Attention is All You Need', completely replaced recurrence and convolutions with attention mechanisms. It consists of an encoder stack and a decoder stack, each made up of identical layers. This architecture became the foundation for BERT, GPT, T5, and virtually every state-of-the-art NLP model today.

2. Encoder Architecture

The encoder is composed of N identical layers (typically N=6). Each layer has two sub-layers: a multi-head self-attention mechanism, and a position-wise fully connected feed-forward network. Each sub-layer has a residual connection followed by layer normalization.

```
LayerOutput = LayerNorm(x + Sublayer(x))
```

The residual connections are crucial — they allow gradients to flow directly through the network without passing through the attention or FFN transformations, making it much easier to train deep networks.

3. Positional Encoding

Since self-attention has no inherent notion of order (it's permutation equivariant), positional encodings are added to the input embeddings to inject information about token positions. The original Transformer used sinusoidal positional encodings:

```
PE(pos, 2i) = sin(pos / 10000^(2i/d_model))
PE(pos, 2i+1) = cos(pos / 10000^(2i/d_model))
```

Sinusoidal encodings have the nice property that the encoding for position pos+k can be expressed as a linear function of the encoding at pos, which may help the model generalize to unseen sequence lengths. Many modern models replace these with learned positional embeddings or relative positional encodings (RoPE, ALiBi).

4. Scaled Dot-Product Attention — Step by Step

Here's exactly how scaled dot-product attention works:

Step	Operation	Shape
1	Project input X into Q, K, V matrices	(seq_len, d_model) → (seq_len, d_k)
2	Compute dot product: Q × K^T	(seq_len, seq_len)
3	Scale by 1/sqrt(d_k)	(seq_len, seq_len)

4	Apply optional mask (for decoder)	(seq_len, seq_len)
5	Apply softmax row-wise	(seq_len, seq_len) — rows sum to 1
6	Multiply by V	(seq_len, d_v)

5. The Decoder and Masked Attention

The decoder has three sub-layers per block: a masked multi-head self-attention layer, a cross-attention layer (attending to the encoder output), and a feed-forward network. The masking in the first sub-layer prevents the decoder from attending to future positions — this is critical for autoregressive generation where the model must predict one token at a time without 'cheating' by looking at future tokens.

6. Feed-Forward Networks in Transformers

Each Transformer layer contains a position-wise FFN applied independently to each position:

```
FFN(x) = max(0, x*W_1 + b_1) * W_2 + b_2
```

Typically the inner dimension d_ff = 4 * d_model (e.g., 2048 when d_model=512). This FFN is where much of the model's 'knowledge' is believed to be stored. Some research suggests FFN layers act as key-value memories.

7. Transformer Hyperparameters (Original Paper)

Hyperparameter	Base Model	Large Model
d_model (embedding dim)	512	1024
Number of layers (N)	6	6
Number of heads (h)	8	16
d_k = d_v	64	64
d_ff (FFN inner dim)	2048	4096
Dropout rate	0.1	0.3
Parameters	65M	213M

8. Why Transformers Beat RNNs

RNNs process sequences token by token, making parallelization impossible during training. LSTMs mitigate vanishing gradients but still struggle with very long sequences. Transformers process all tokens simultaneously, support full parallelism on GPUs/TPUs, handle long-range dependencies with O(1) path length between any two positions, and scale remarkably well with data and compute — leading to the era of large language models.