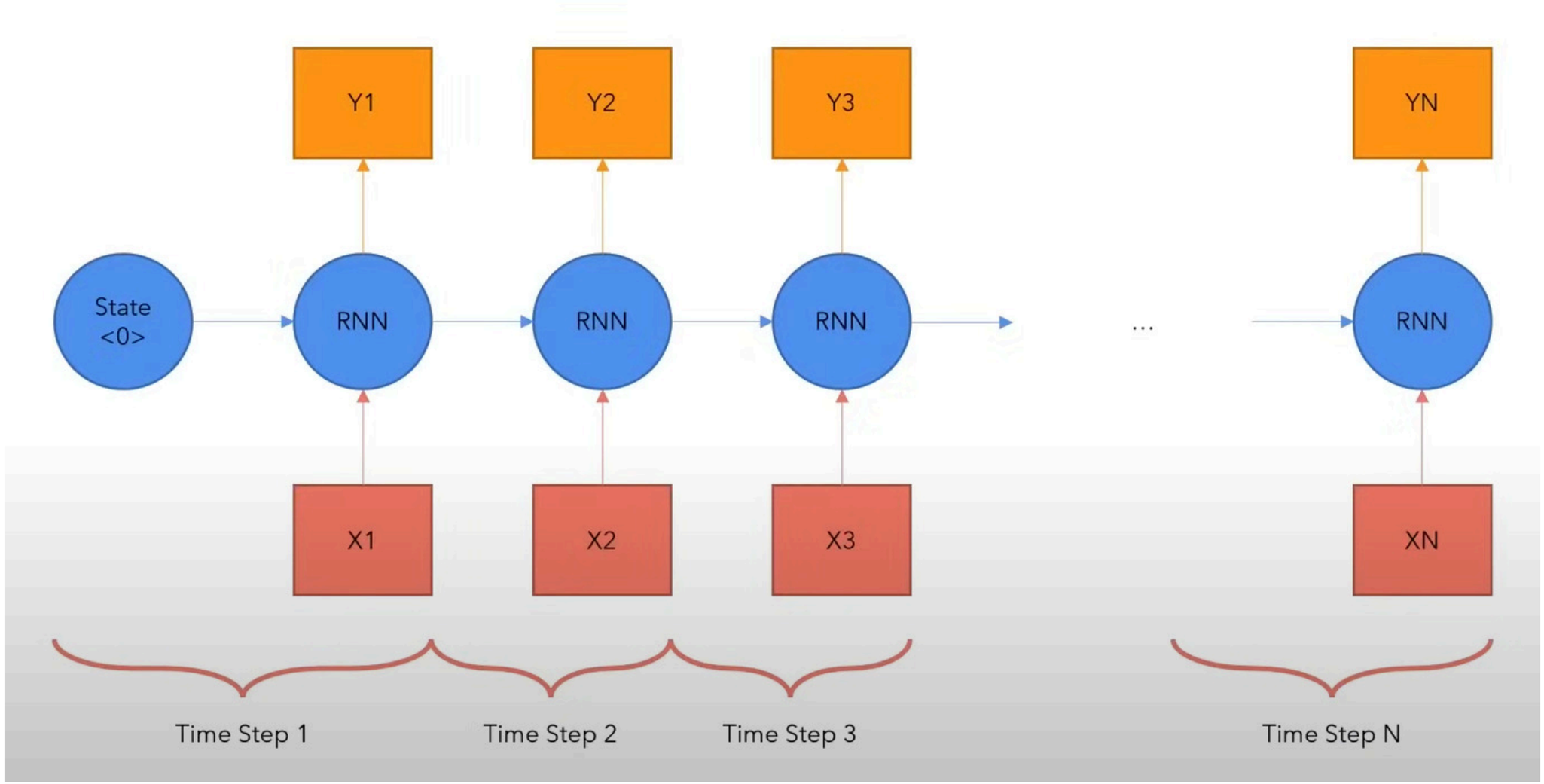
Transformers in LLM

Recurrent Neural Network

An **RNN (Recurrent Neural Network)** is a type of neural network designed to handle sequential data by maintaining a hidden state that captures information from previous time steps using Tanh nonlinearity.

1. It processes sequences step-by-step, using the current input and the previous hidden state to produce the current output and update the hidden state.
2. However, it struggles with long-term dependencies due to vanishing gradients.
3. Despite its simplicity, RNNs are the foundation for more advanced architectures like LSTMs and GRUs.

Recurrent Neural Network (RNN)



[Attention is all you need (Transformer) - Model explanation (including math), Inference and Training]

What is a Transformer?

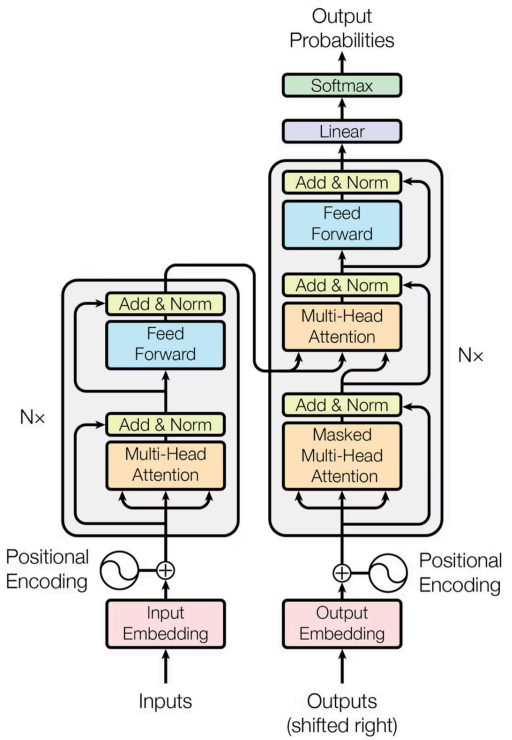


Figure 1: The Transformer - model architecture.

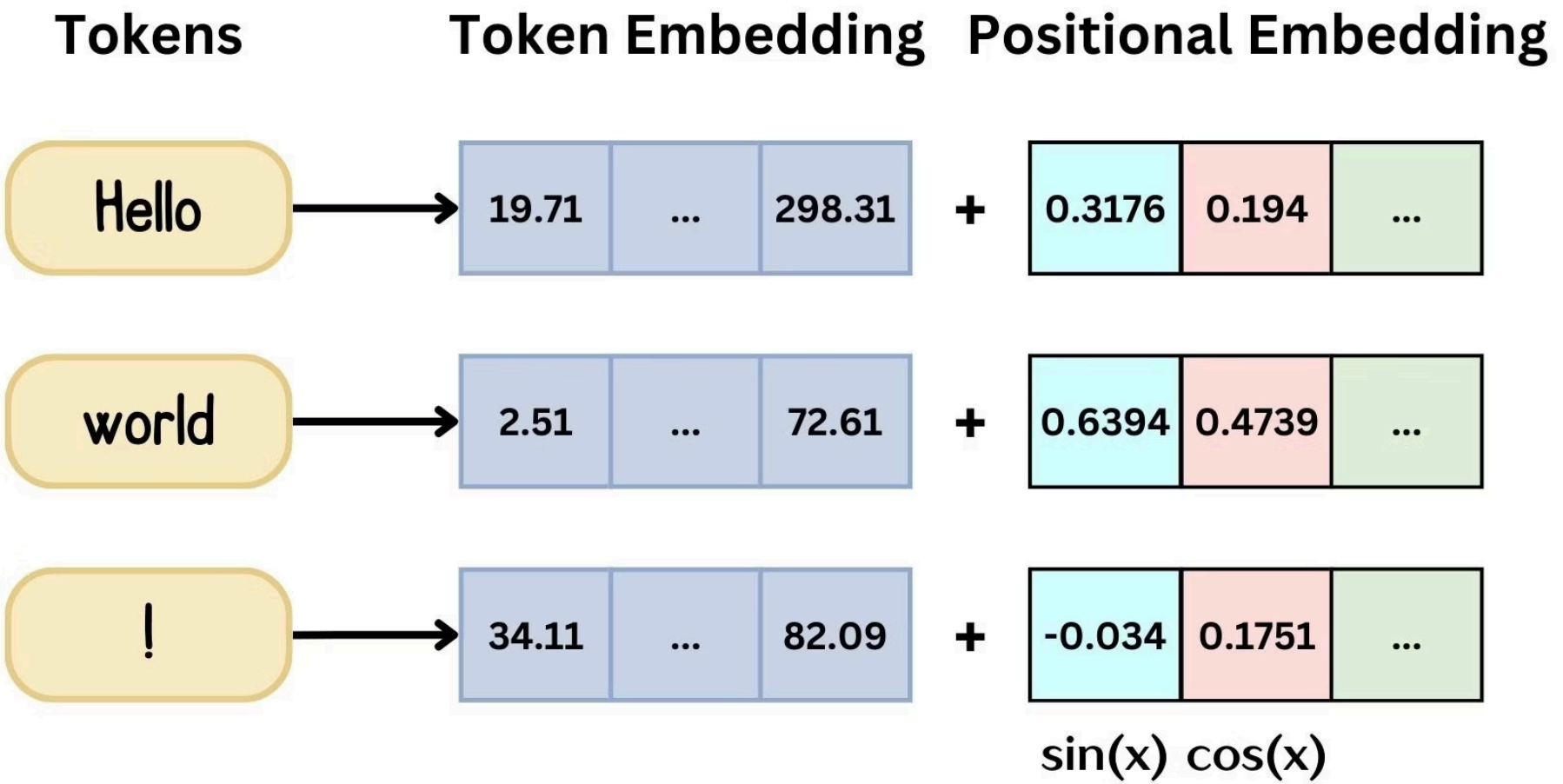
- The Transformer is a deep learning model introduced in 2017 by Google Brain in the paper "Attention Is All You Need".
- Transformers rely entirely on an attention mechanism to draw global dependencies between input and output.
- It processes sequences of data, such as text, by considering all parts of the sequence simultaneously.
- Enabling highly parallelized training and improved performance on long sequences.

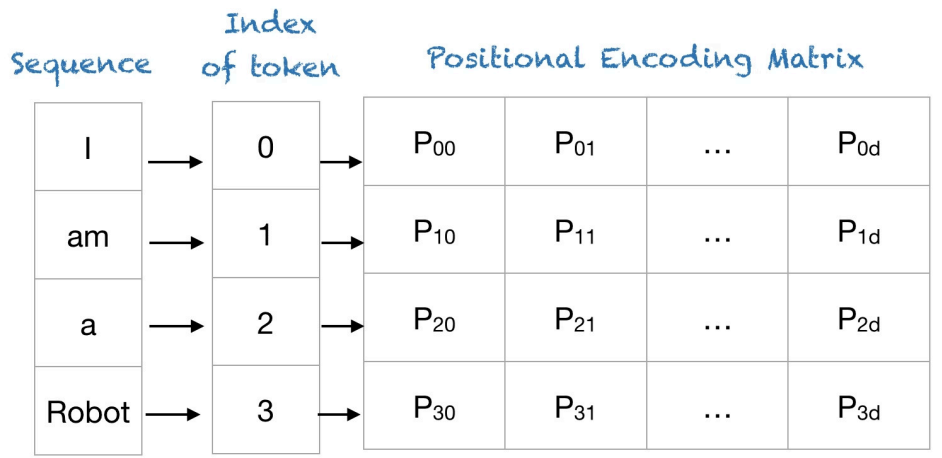
[Attention Is All You Need]

Why Transformers? Overcoming RNN and LSTM Limitations

RNNs and LSTMs read one word at a time, which:

- Makes them **slow** due to sequential processing
- The gradients vanishes or exploding gradient due to non-linearity
- Struggles with **long-term dependencies** (forgetting context from earlier), and also vanishing or exploding gradient.
- Transformers introduced **attention** to directly model relationships between all words.
- **Parallelizable**, making it ideal for training on GPUs.





Positional Encoding Matrix for the sequence 'I am a robot'

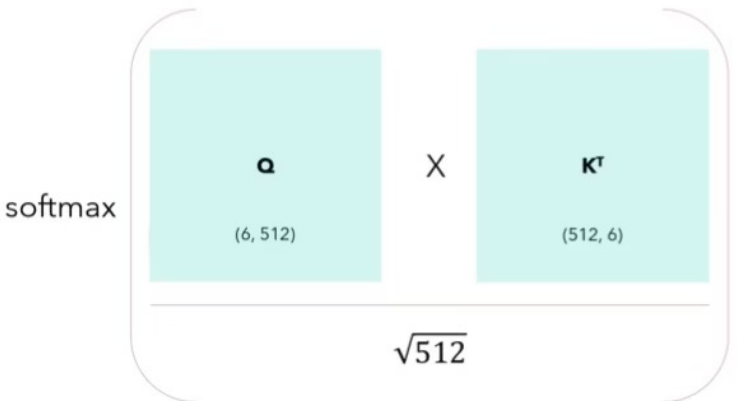
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

Overall Architecture: Encoder Structure

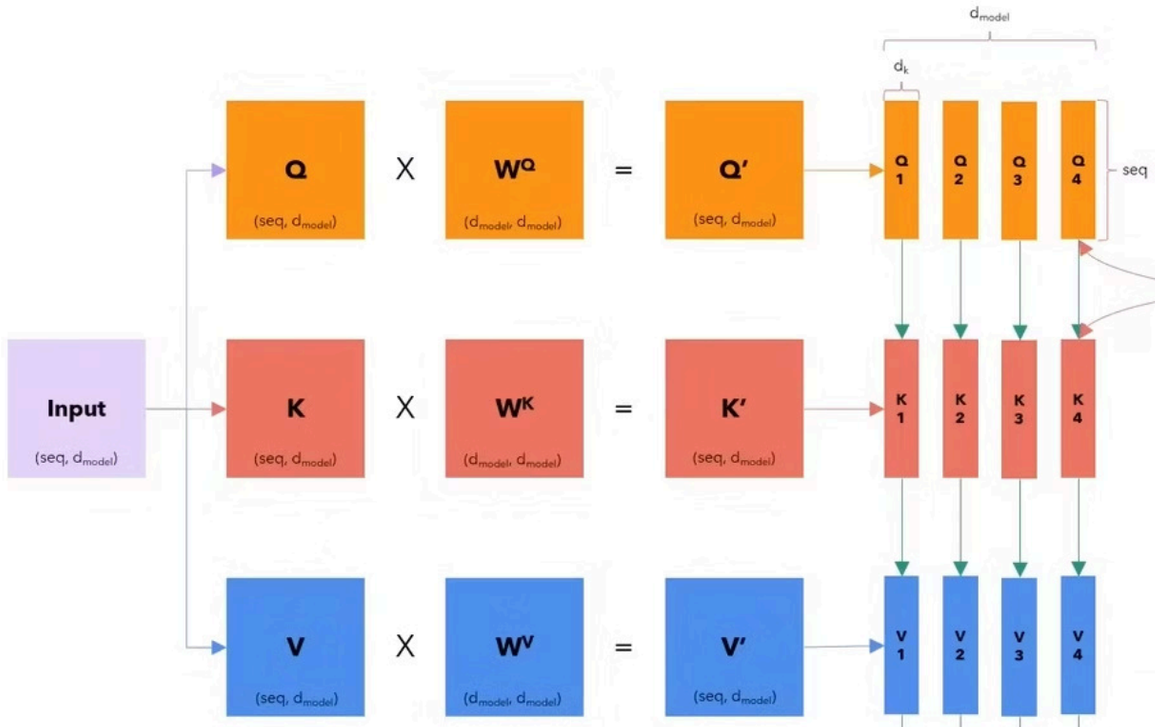
- Convert the input sequence into tokens using a tokenizer
- Map each token to a vector using a learnable embedding matrix
- Add positional encoding to each token embedding to include word order and position in the sequence
- The result is the final embedding passed into the Transformer model

The Self-Attention Mechanism

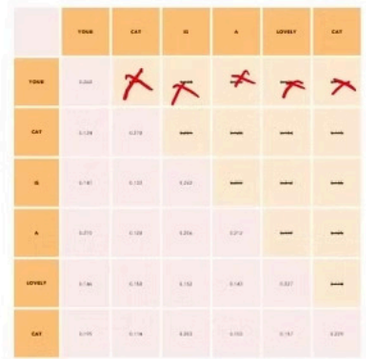


- For each token, create Query, Key, and Value vectors using learned weight matrices
- Compute attention scores by taking the dot product of the Query with all Keys
- Apply softmax(Q, K,) to turn scores into attention weights
- Multiply the attention weights with the Value vectors and sum them to get the output for that token
- In self attention, Key and query are transpose of each other

Multi-Head Attention: Diverse Perspectives



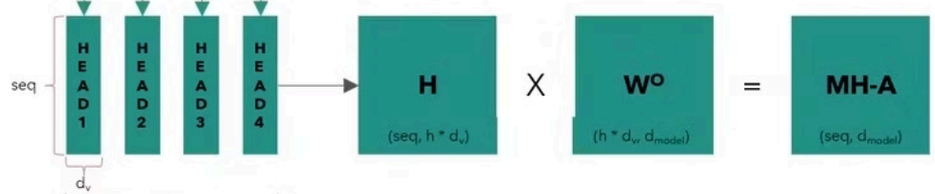
$$softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) =$$



All the values above the diagonal
are replace with **-∞** *before* applying the softmax,
which will replace them with zero.

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$



Transformer vs. RNN/LSTM: Comparison

Faster

Training Speed

Transformers' parallelizable attention mechanism significantly accelerates training compared to sequential RNN/LSTM models.

Better

Long-Range Dependencies

The attention mechanism excels at capturing relationships between distant elements in sequences, a common challenge for RNNs and LSTMs.

Superior

Performance

Transformers consistently achieve state-of-the-art results across a wide range of sequence-to-sequence tasks, including machine translation and text generation.

References:

- Attention is all you need by Google.
- Attention is all you need (Transformer) - Model explanation (including math), Inference and Training