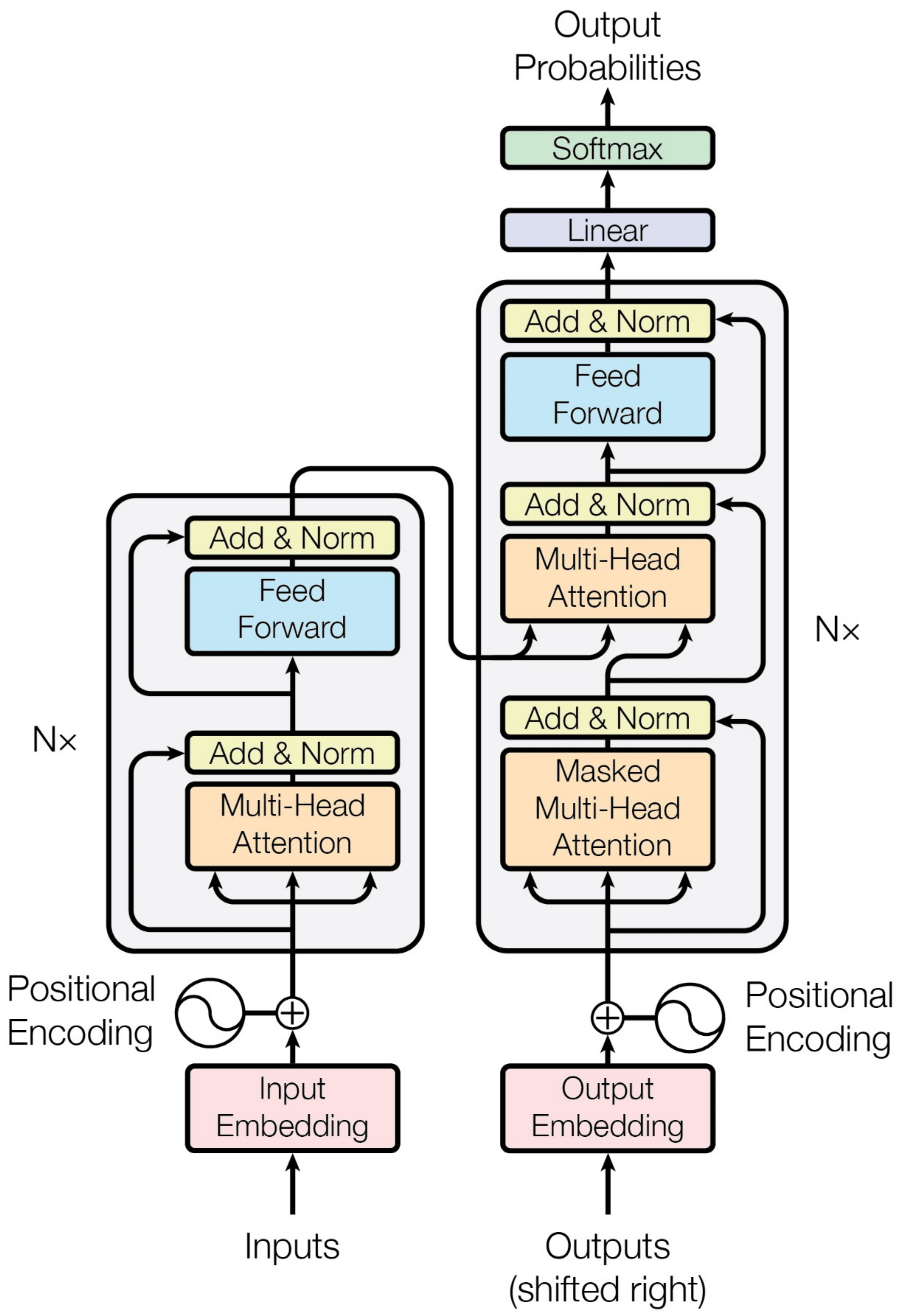
Attention and the Transformer Architecture

Our work: FlashAttention-2 on Hopper

- Optimized **Multi-Head Attention** (MHA) via kernel fusion techniques on **NVIDIA Hopper™ architecture**.
- Attention formula is: $O = \mathrm{softmax}\left(1/\sqrt{d}\ QK^T\right) V$, where Q, K, and V are (N x d)-matrices (*query*, *key*, and *value*).
- Base algorithm is Tri Dao's **FlashAttention-2**.
- Goal today: Explain context of optimization work in the landscape of deep learning models and the transformer model architecture.

Transformer architecture

Figure 1 from
"Attention Is All You Need"
Vaswani et. al., 2017



Transformers – origin story

- Transformers were introduced to address limitations of recurrent neural networks (RNNs) and variants (LSTM, GRUs) in sequence processing tasks.
- RNNs process data sequentially, but to extend models to large sequence lengths in a computationally efficient way we want to exploit parallelization.
- At the time, the *attention* mechanism was introduced in conjunction with RNNs to model large-scale dependencies within data.
- A key insight of "Attention Is All You Need" was to dispense with recurrence and rely entirely on attention.

Transformer model, initial steps

- **Tokenization**: raw input data converted into a batch of tokens.
  - Has dimension (B = batch size, N = sequence length), e.g. B = 4, N = 4096.
- **Input embedding**: tokens converted into vectors of length D = embedding dimension, e.g. D = 2048.
  - Yields tensor of dimension (B, N, D).
- **Positional encoding**: used to know about order of the sequence.
  - Adds vectors to initial token embeddings that encode positional info.

Transformer model, apart from attention

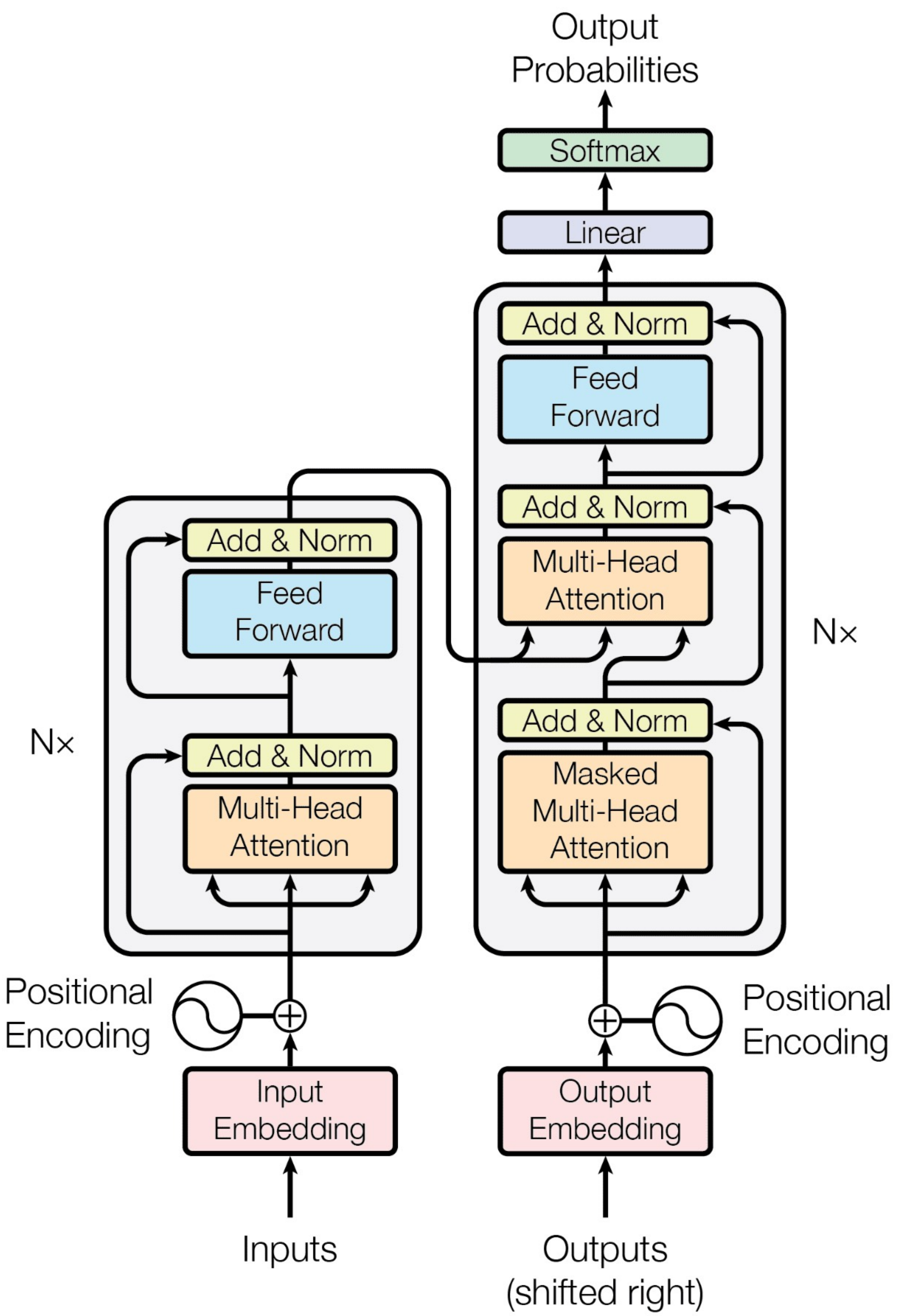
- Attention sublayer *transforms* the tensor of shape (B, N, D).
- Output of attention then passed through **feedforward neural network**.
    - Involves two linear transformations and a ReLU activation in between.
    - Have $\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$, with learnable parameters.
- Also apply **layer normalization** and **residual connections** to outputs.
    - Have $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$. Sublayer is attention or FFN.
- Stack identical copies of these layers (attention + feedforward), e.g. 6 copies.

Self-Attention and Multi-Head Attention

- We saw the attention formula $O = \mathrm{softmax}\left(1/\sqrt{d}\ QK^T\right)V$.
- *Q*, *K*, *V* matrices arise from *learnable projections* of the input tensor.
    - $Q = XW^Q, K = XW^K, V = XW^V$. Weights are learned.
- In multi-head attention with H heads, have a set of H many triples of these projection matrices. (H divides embedding dimension D, and D = H*d).
- Then have formula: $\mathrm{MultiHead}(X) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_H)W^O$ for $\mathrm{head}_i = \mathrm{Attention}(XW_i^Q,\ XW_i^K,\ XW_i^V)$.

Figure 1, revisited

- We discussed layers in the encoder.
- Decoder: each layer has three sublayers. First MHA is masked. Insert another MHA that receives output of encoder stack.
- Apply softmax+linear to output of decoder stack. This makes the token prediction.
- Variants: encoder-only (BERT), decoder-only (GPT), and more!



Unpacking the attention formula

- Why *query*, *key*, *value* in $O = \text{softmax}\left(1/\sqrt{d}\ QK^T\right)V$?
  - **Query**: current item for which we're computing the attention weights.
  - **Key**: the items in the sequence compared against the query.
  - Comparison done using **scaled dot product**.
  - **Softmax** is smooth approximation to argmax: think of it as selecting the largest entry in the vector, but in a way suitable for doing backprop in training.
  - **Value**: finally, use attention weights to create a weighted sum of values per every query item.

References

- "Attention Is All You Need". Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. https://arxiv.org/abs/1706.03762.

- Author order randomized in the paper with equal contribution.

- There are many, many introductions to this famous paper. We found the following tutorial helpful: https://jalammar.github.io/illustrated-transformer/.