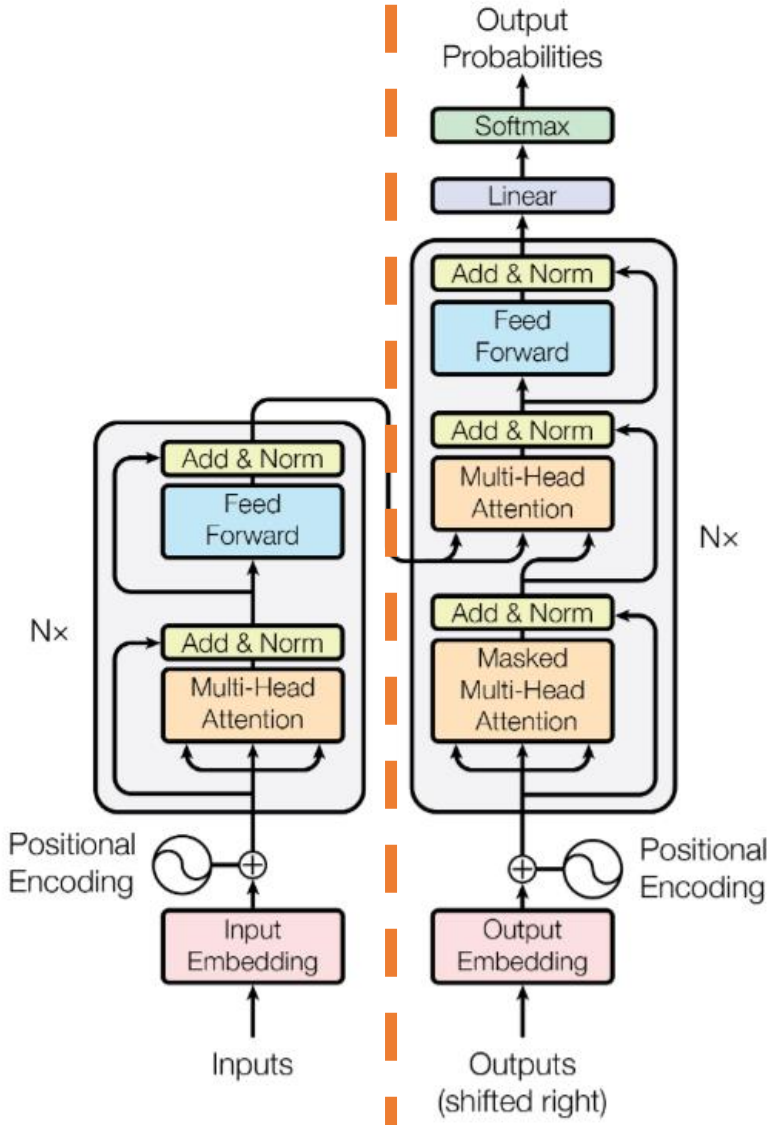
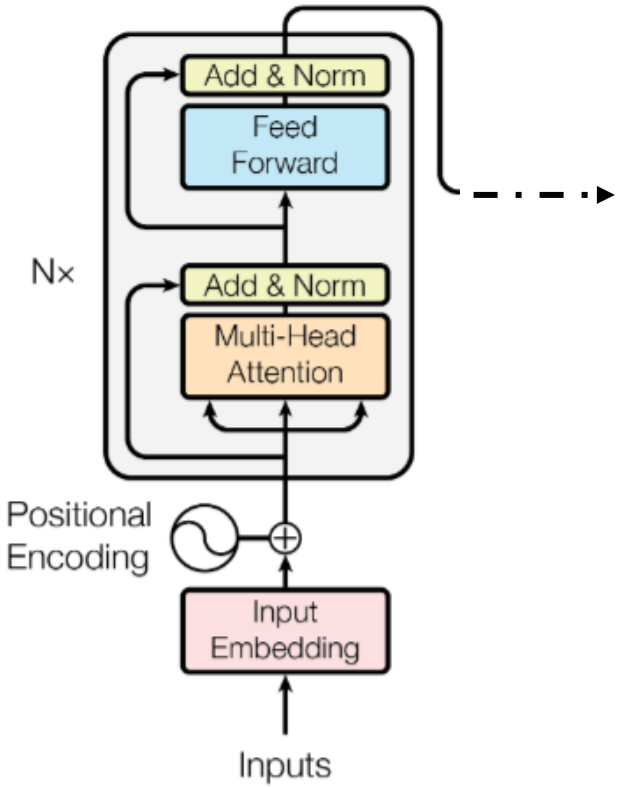
Transformers

Encoder



Decoder



Input Embedding

A lookup table with learned parameters. The parameters are initialized randomly. Maps one-hot encodings of words in vocabulary to vectors of dimension d_{model}. Consult the nn.Embedding class in PyTorch.

Positional Encoding

Without this addition, the encoder's output embedding would be invariant to the order of the input sequence. The positional encoding is a vector of dimension d_{model}.

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

Feed Forward

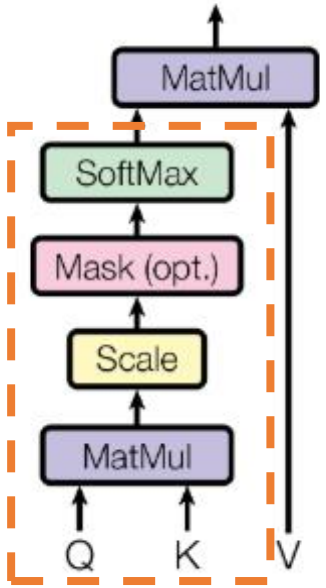
Nothing unusual here. Two linear transformations and a *ReLU* activation in between. $d_{in} = d_{out} = d_{model}$. With a brief enlargement to $d_{ff} = 2048$ after the first layer.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Add & Norm

The model uses residual connections, so we need to add these back. Then we perform normalization, specifically layer normalization. We cannot perform batch normalization with varying length sequences

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$



In the encoder, we perform **self-attention (aka bidirectional attention)**. The query, keys and values all come from the input sequence.

MatMul

Nothing unusual here, multiply the query matrix by the key matrix. Resulting matrix will have dims $d_{seq} \times d_{seq}$

Scale

Every value in the resulting matrix is scaled by $\sqrt{d_k}$

Mask (opt.)

Mask attention weights according to a Boolean matrix, set True entries to $-1e8$

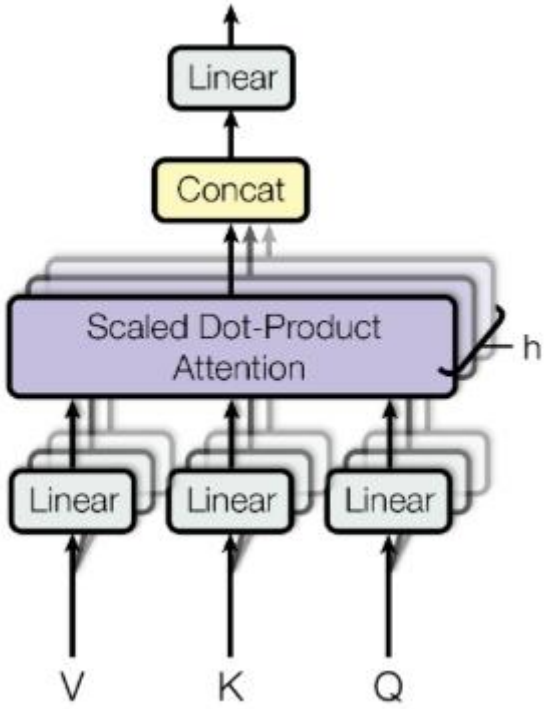
SoftMax

Apply Softmax to the rows of the matrix

MatMul

As above, no tricks here. Multiply the attention weights by the value matrix. Dim to $d_{seq} \times d_v$

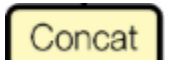
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



In multi-head attention, we repeat the sequence of operations from the previous slide h times.

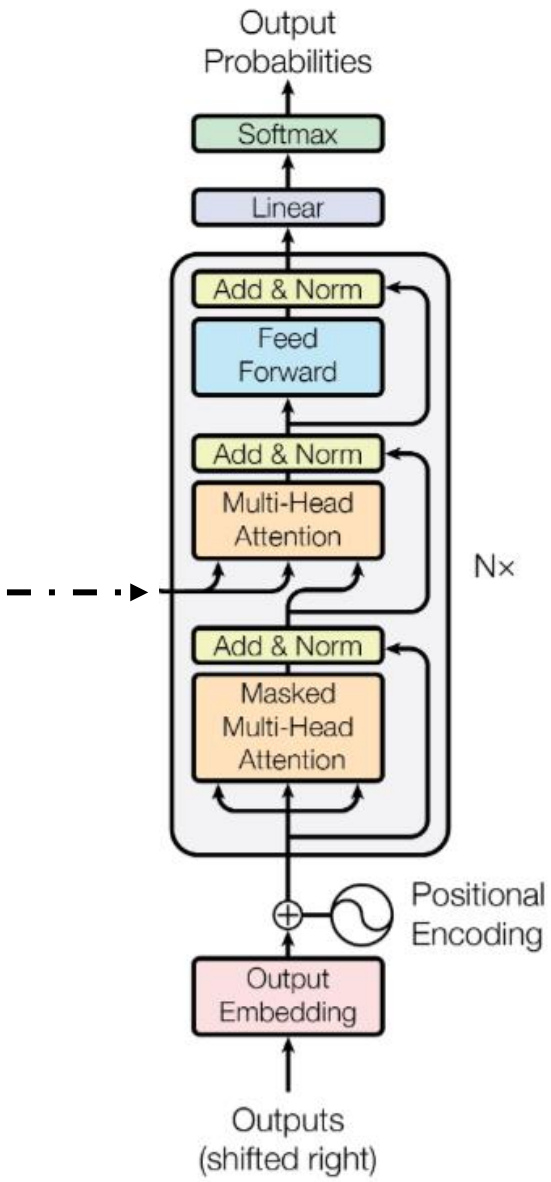
Each head $i \in 1, \ldots, h$ has its own triplet of weight matrices, W_Q^i, W_K^i, W_V^i of dimension, $d_{model} \times d_k$, $d_{model} \times d_k$, $d_{model} \times d_v$ (respectively), which are applied to the input sequence.

Apply scaled dot-product attention on the linearly transformed query, key and values for each head.



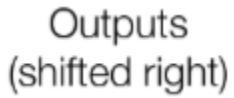
The output of each head is concatenated. This results in an output of dimension $d_{seq} \times (h * d_v)$

A final linear layer is applied to the concatenated outputs, the weight matrix for this layer $W^o \in \mathbb{R}^{hd_v \times d_{model}}$ brings our embeddings back to d_{model}.



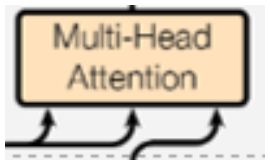
During training, the model uses *teacher-forcing,* it uses the correct output from the dataset as the input to the decoder. This means that decoding can be parallelised during training.

At evaluation time the decoder generates the output sequence one element at a time.

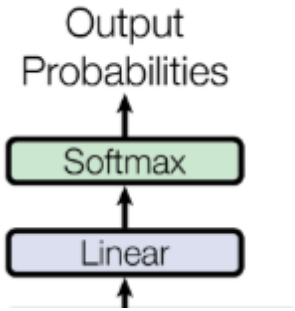


The output sequence, *(shifted right),* we begin with a start token.

As for the encoder, with the caveat that we use the masking of attention weights to 'hide' future elements of the output sequence from the model (during training).



Once again, fundamentally the same as previously but this time the keys and values are the embedded input sequence, and the queries are the decoder output so far.



The final task of the decoder is to turn the embedding so far into the right form for the task. In the paper this task is a sequence-to-sequence translation, therefore the decoder outputs a probability over the vocabulary for each element of the output sequence.

Advantages of Transformers

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

For short sequences, n^2 complexity is better than d^2 complexity (for large sequences there are some tricks)

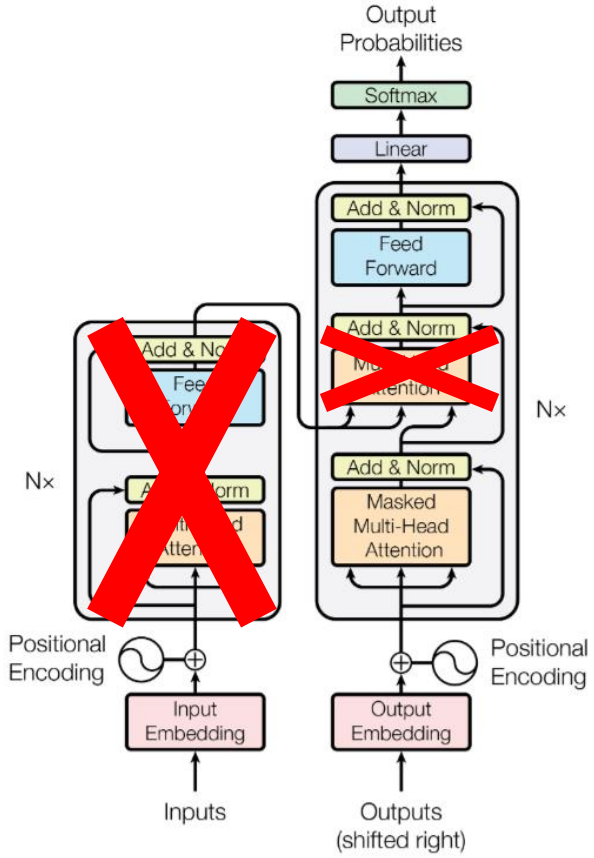
The maximum path length between sequence elements is constant.

If we desire it, we can make transformers invariant to the order of the sequential input.

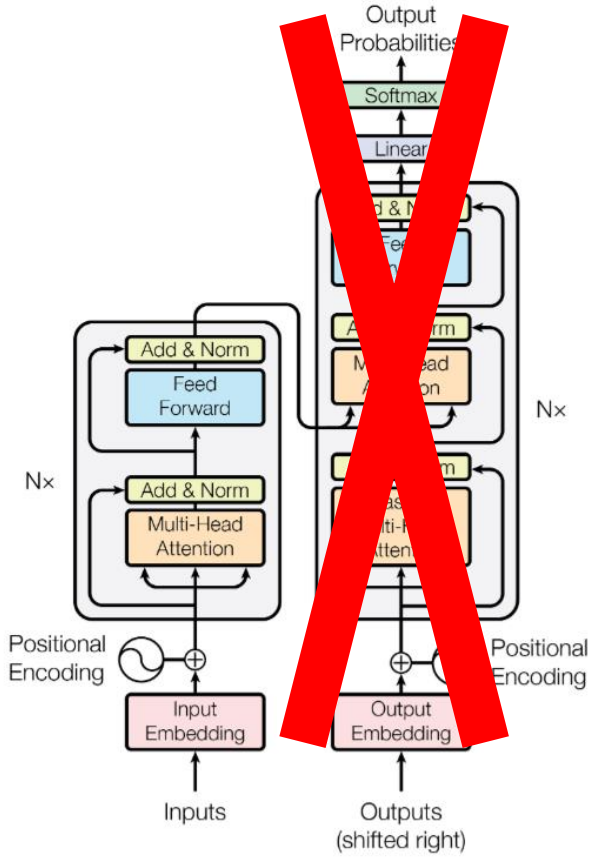
Transformers are highly parallelizable (This comes at the cost of large memory requirements)

Other Transformer Architectures

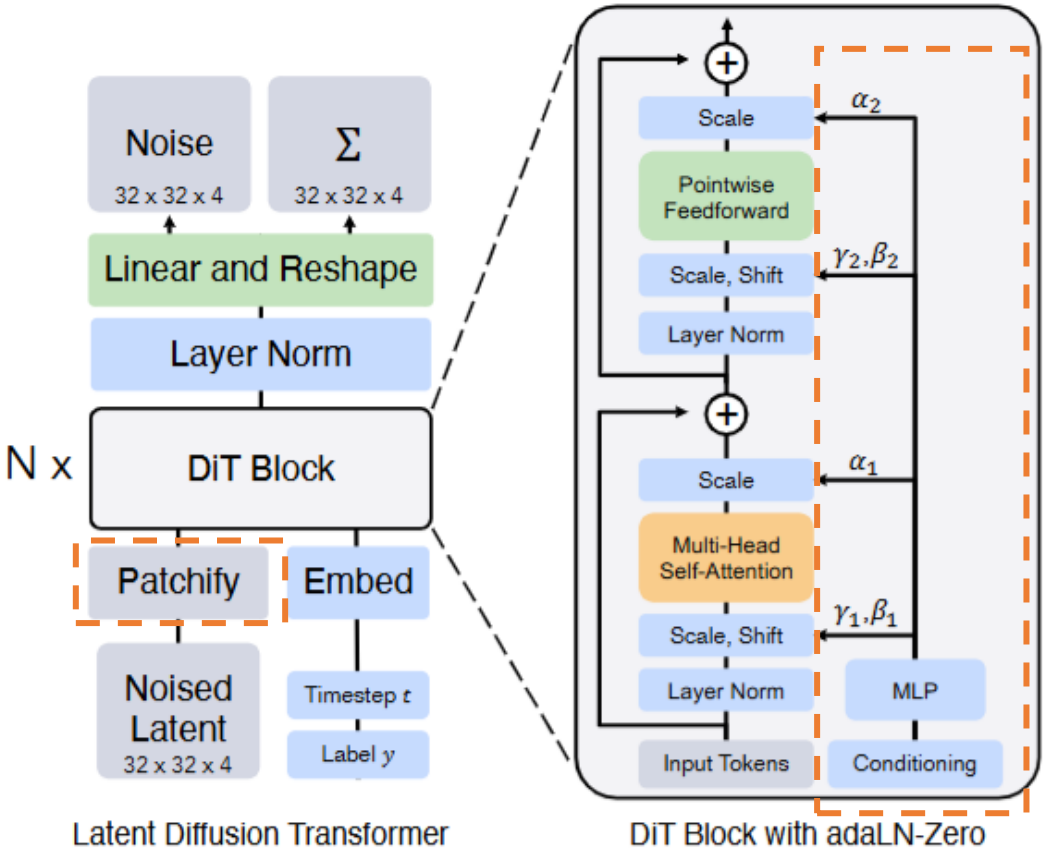
Decoder Only Transformer (Text Generation) (GPT)

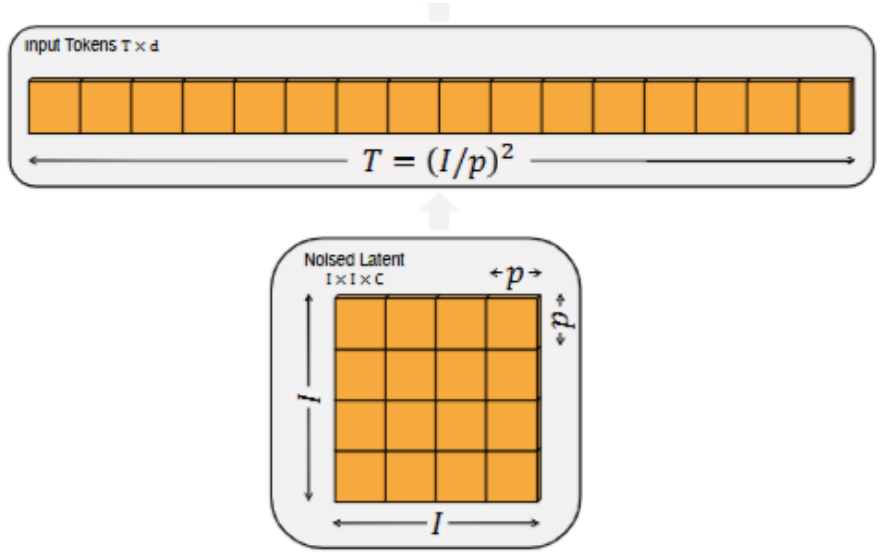


Encoder Only Transformer (Sentiment Analysis) (BERT)



Transformers in Diffusion Models





[Diffusion Transformer architecture]

Recommended Further Reading

- The original paper – “Attention is All You Need” – Link
- A technical report produced by DeepMind giving a more comprehensive overview of the mathematics of Transformers with pseudocode – Link
- A blog post explaining Transformers with many animated diagrams explaining each component – Link
- Talk by one of the authors of Attention is All You Need - Link