

# Transformers

## Background and Construction

Shounak Shirodkar

MINRO, IIIT-Bangalore

May 2022

# Table of Contents

## 1 Introduction

- Background
- Prerequisites
- Recap

## 2 Transformers

- The need for Transformers
- The Transformer architecture
- Attention
- Position-wise Feedforward Neural Nets
- Embeddings and Softmax
- Positional Encoding

# Background

- 1 RNNs were first proposed in [Rumelhart et al., 1986].
- 2 LSTM networks were proposed in [Hochreiter and Schmidhuber, 1997]. Around 2007, the use of LSTM in speech recognition was pioneered, where it outperformed traditional models in certain applications.
- 3 Older seq2seq models had limitations with large training data due to sequential processing.
- 4 A workaround, the Transformer model, was proposed in [Vaswani et al., 2017], which processes all input data “at once”, and allows greater parallelization. The Transformer takes far less time to train, and is also able to handle larger training data.

# Prerequisites

- Neural Networks (often shortened to Neural Nets)
- Recurrent Neural Nets
- Long Short-Term Memory
- Residual Neural Nets
- Layer Normalization
- Attention

**Neural Nets**, Artificial Neural Networks (ANNs), or simply neural nets (NNs) are computing systems inspired by biological neural networks. Modelled as a network of nodes, where a node corresponds to a neuron while the connecting edge is analogous to a neural synapse. In its simplest form, an ANN can be represented as a feedforward NN. NNs have several applications, however we shall only study their application in time-series prediction and modeling.

**Recurrent Neural Nets**, or RNNs, are a class of NNs where connections between nodes form a *temporal* sequence. RNNs are Turing complete [Hytyniemi, 1996], and can process variable-length inputs. RNNs are directed cyclic graphs with infinite impulse response, as opposed to convolutional neural nets (CNNs) which have a finite impulse response, and are directed acyclic graphs.

**Long Short-Term Memory** RNNs, or LSTMs, are neural nets with both feedforward and feedback connections. Simple RNNs use backpropagation, which risks *vanishing* or *exploding* gradients due to finite precision. LSTMs allow gradients to flow unchanged, and partially resolve the vanishing gradient problem.

**Residual Neural Nets**, or ResNets, are deep feedforward NNs with *skip* connections, that jump over some layers. Like in LSTMs, ResNets are advantageous in mitigating both, vanishing gradients, and the accuracy saturation problem.



**Layer Normalization** [Ba et al., 2016] estimates normalization statistics from the summed inputs to the neurons within a hidden layer, so that the normalization does not introduce any new dependencies. Mathematically

$$\mu^l = \frac{1}{H} \sum_{i=1}^H a_i^l$$
$$\sigma^l = \sqrt{\frac{1}{H} \sum_{i=1}^H (a_i^l - \mu^l)^2}$$

where  $a^l$  represent the vector of the summed inputs to the neurons belonging to layer  $l$ , and  $H$  is the number of hidden units. There is no restriction on mini-batch size, so this can be used in the pure online regime with batch-size 1, unlike batch normalization.

**Attention** is a technique used within neural nets that corresponds to cognitive attention, attempting to lend greater focus to the more important parts of the data while ignoring the noise.

An attention function maps a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

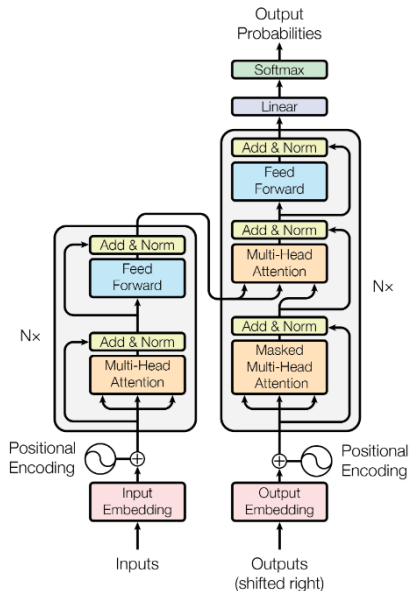
Attention mechanisms have been used to allow modeling of dependencies without regard to proximity, in conjunction with RNNs.

# The need for Transformers

- Recent advances in parallel computation on GPUs make parallelizable algorithms favorable.
- Recurrent models such as RNNs and LSTMs require sequential computation due to the sequence of hidden states generated from training data. This becomes a critical factor at longer sequence lengths due to memory constraints.
- Despite recent improvements in computational efficiency, the fundamental constraint of sequential computation remains.

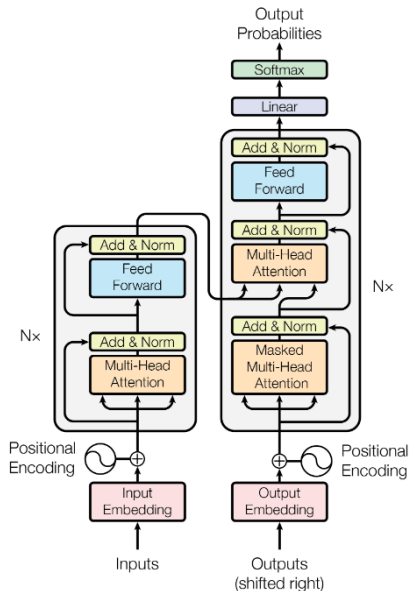
The Transformer avoids recurrence, instead depending completely on the attention mechanism, and allows significantly more parallelization.

# The Transformer architecture



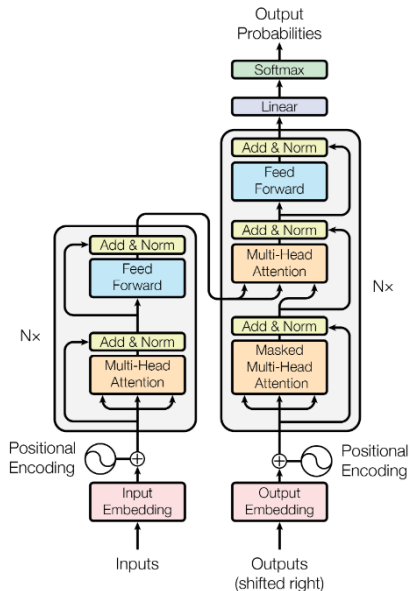
Like in most competitive deep learning transduction models, Transformer consists of two larger components: the **encoder**, and the **decoder**.

# The Transformer architecture



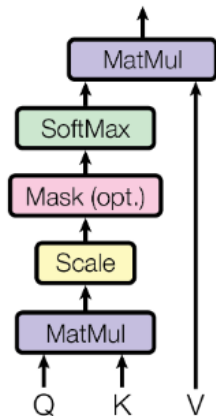
The **encoder** has  $N = 6$  identical layers. Each layer has 2 sub-layers: a multi-head self-attention mechanism, and a simple, position-wise fully connected feedforward neural net. A residual connection is made around each sub-layer, followed by layer normalization.

# The Transformer Architecture



The **decoder** has  $N = 6$  identical layers. Each layer has 3 sub-layers: an additional sub-layer performing multi-head attention over the output of the encoder stack is introduced. Residual connections are deployed around each sub-layer, followed by layer normalization. The output embeddings are offset by one position, and the self-attention sub-layer masks subsequent positions to maintain sequential learning.

# Scaled Dot-Product Attention



$Q \rightarrow$  queries

$K \rightarrow$  keys

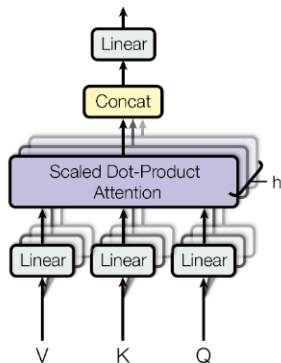
$V \rightarrow$  values

Let  $\dim(K) = d_k$  and  $\dim(V) = d_v$ , then define the attention function as

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

Additive attention and dot-product attention are similar in complexity, but the latter is faster and space-efficient. The former outperforms simple multiplicative attention for larger  $d_k$ . By scaling the dot-product by  $1/\sqrt{d_k}$ , we avoid the regions of softmax with small gradients.

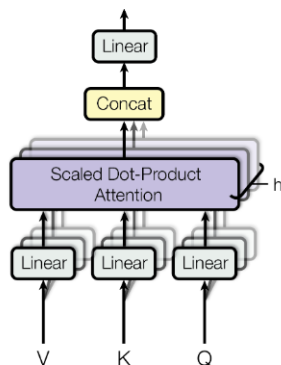
# Multi-Head Attention



$Q$ ,  $K$  and  $V$  are linearly projected  $h$  times with different, learned, linear projections to  $d_k$ ,  $d_k$  and  $d_v$  dimensions respectively. The attention function is then applied to these projected versions of  $Q$ ,  $K$  and  $V$  in parallel, returning  $d_v$  dimensional outputs, which are concatenated and once again projected, resulting in the final values.



# Multi-Head Attention



Multi-head attention is defined as

$\text{MultiHead}(Q, K, V) =$

$$\text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O$$

where

$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ ,  
and the projections are  $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$ ,  
 $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ ,  $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$  and  
 $W^O \in \mathbb{R}^{h d_v \times d_{\text{model}}}$ . In [Vaswani et al., 2017],  
 $h = 8$  and  $d_k = d_v = d_{\text{model}}/h = 64$ .

# Position-wise Feedforward Neural Nets

Every layer, in both the encoder and the decoder, contains an identical position-wise feedforward neural net consisting of two linear transforms with an ReLU activation function in between. Network parameters vary from layer to layer, even though the linear transforms are identical. Input and output dimensions are  $d_{model} = 512$ , and the inner layer has dimensionality  $d_{ff} = 2048$ .

# Embeddings and Softmax

The raw input and output tokens passed during training phase are converted to vectors of dimension  $d_{model}$  via learned embeddings. Learned linear transform and softmax are used convert decoder output to predict next-token probabilities. The Transformer shares the same weight matrix between the two embedding layers and for the linear transform (prior to the softmax at the top of the stack). In the embedding layers, the weights are scaled by  $\sqrt{d_{model}}$ .






# Positional Encoding

Since there is no recurrence or convolution, positional encoding is used to help the model understand the sequential order. These encodings are added to the input embeddings at the bottom of the stack(s). The scheme used for the Transformer model is

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

where  $pos$  is the position in the sequence, and  $i$  is the dimension.

# References

-  Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016).  
Layer normalization.
-  Hochreiter, S. and Schmidhuber, J. (1997).  
Long short-term memory.
-  Hyotyniemi, H. (1996).  
Turing machines are recurrent neural networks.
-  Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986).  
Learning Representations by Back-propagating Errors.
-  Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017).  
Attention is all you need.