Transformers – Lecture 3

Bill Byrne

Lent 2022

Neural Machine Translation and Dialogue Systems – MLMI8

MPhil in Machine Learning and Machine Intelligence

UNIVERSITY OF CAMBRIDGE

Outline

The Transformer

- *Attention is all you need*, Vaswani et al. NIPS 2017
- Architecture

Encoder – Decoder architectures, BERT & GPT-2

Some useful tutorials

- The Illustrated Transformer
- Everything GPT-2: 2. Architecture In-depth
- The Illustrated GPT-2 (Visualizing Transformer Language Models)
- Natural Language Processing with Transformers (the 'Hugging Face book')

Translation via Sequence-to-Sequence Models

Aim: define conditional distributions over target language sentences, given a source sentence

- Source sentence **x** = $x_1 \ldots x_n$
- Target sentence **y** = $y_1 \ldots y_m$,
- Shorthand for the history at step *i* : $\mathbf{y}_{<i} = y_1 \ldots y_{i-1}$

The Transformer defines a conditional probability over target language sequences: $P(\mathbf{y} \mid \mathbf{x}; \theta)$

- this is consistent with symbol-wise computation

$$P(y|x;\theta) = \prod_{i=1}^{m} \underbrace{P(y_i|y_{<i}, x;\theta)}_{\text{Transformer}}$$

In this lecture we'll review the Transformer architecture, training procedures, and related models such as BERT and GPT

Transformer

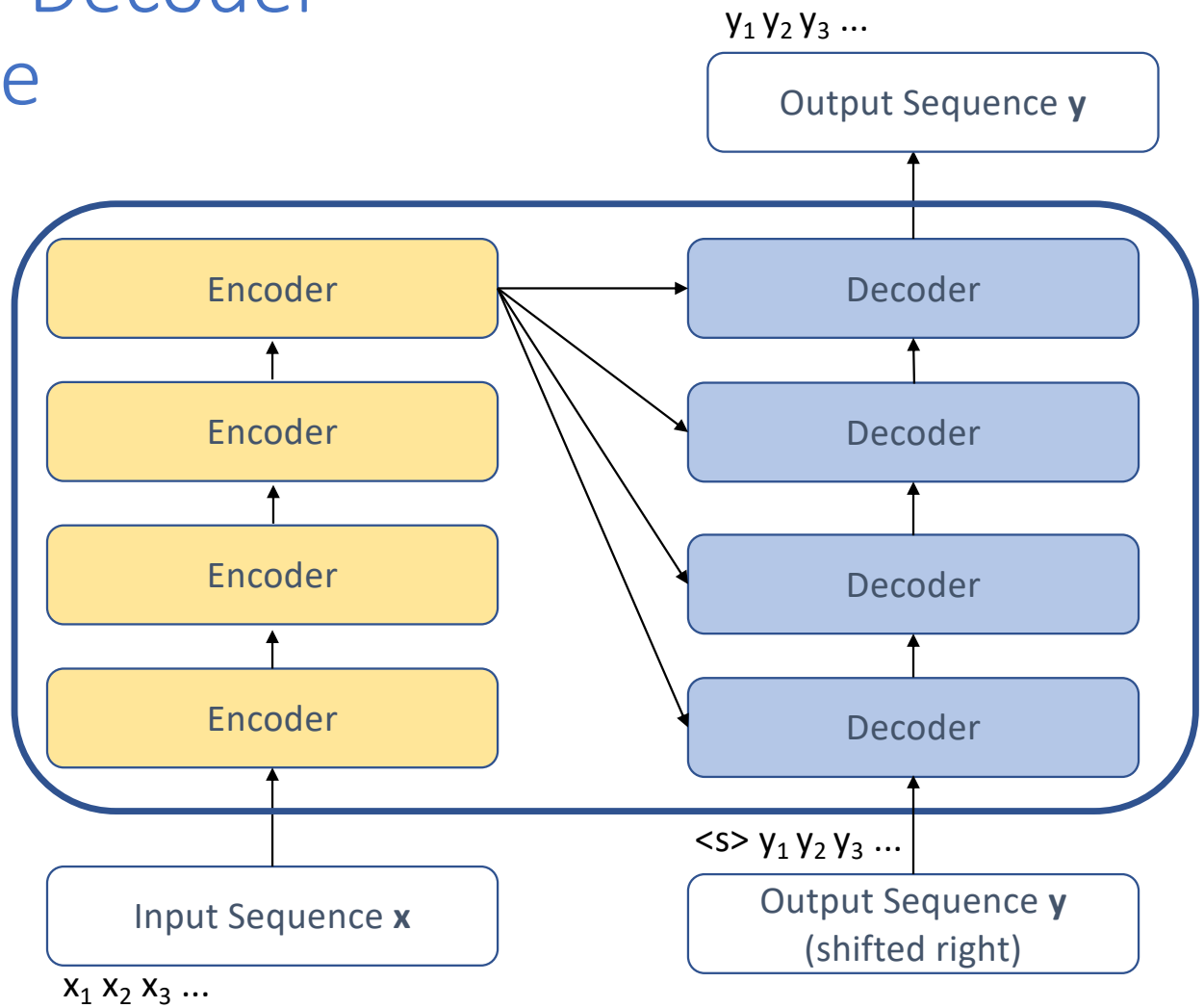
Encoder-decoder structure

- The encoder maps an input sequence of symbol representations to a sequence of continuous representations
- The decoder defines a distribution over sequences, can also be computed one element at a time
  - *causal* dependencies in the decoder structure
- The model is auto-regressive, consuming the previously generated symbols as additional input when generating the next symbol

Avoids recurrence

- **attention mechanism** establishes dependencies between input and output.

Encoder-Decoder Structure



Encoder and Decoder Stacks

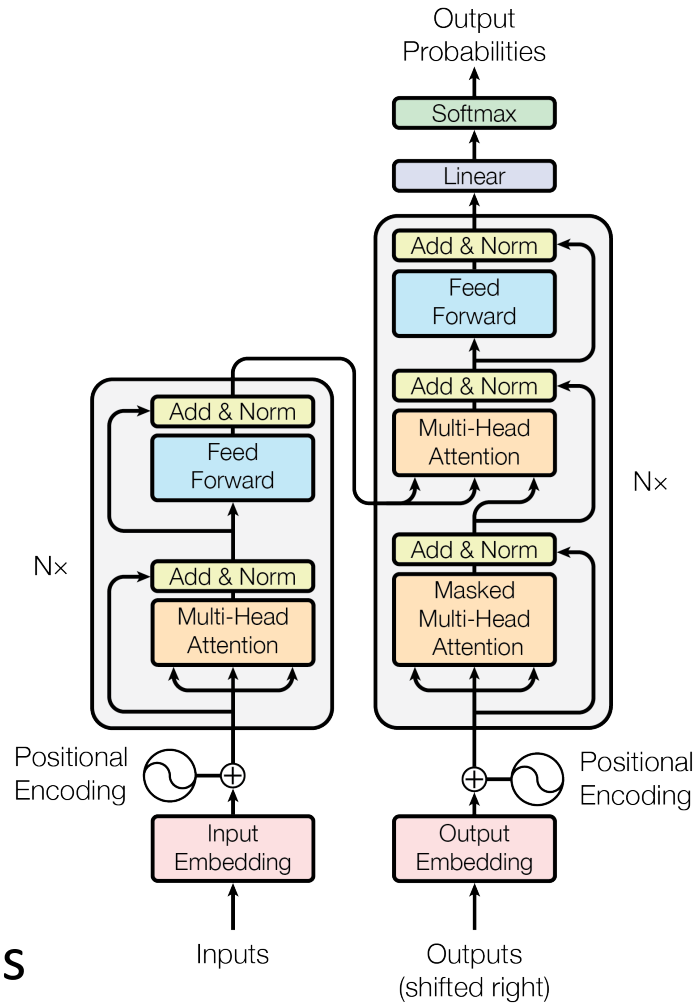
Encoder

- N identical layers
- Multi-head self-attention mechanism
- Position-wise fully connected FFNN
- Residual connection with Layer Normalisation

Decoder

- M identical layers
- Multi-Head Attention over the output of the encoder stack
- Masked Multi-Head Attention

Positional encoding of the input and output embeddings



UNIVERSITY OF CAMBRIDGE

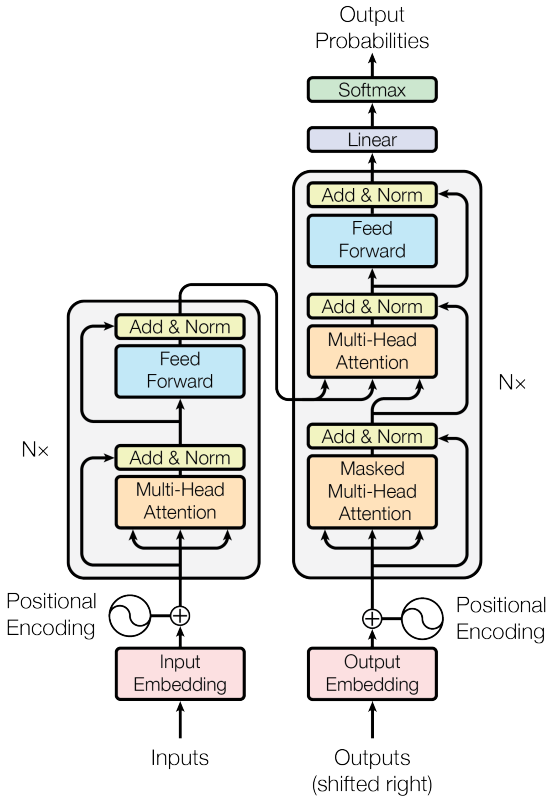
Input and Output Embeddings

How to derive continuous representations for symbols:

Suppose we have a vocabulary $V = \{v_1, \ldots, v_K\}$ of size K

- Notation $[v_i]$ indicates a unit vector in $[0,1]^K$ with a 1 in position i
  - For example, if *hat* is the 17th vocabulary word in V, then

$$[\text{hat}] = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}_{K \times 1} \leftarrow \text{position 17}$$



- Input and output embedding can be done with simple embedding matrices
  - embedding of the word `hat' is W [hat]

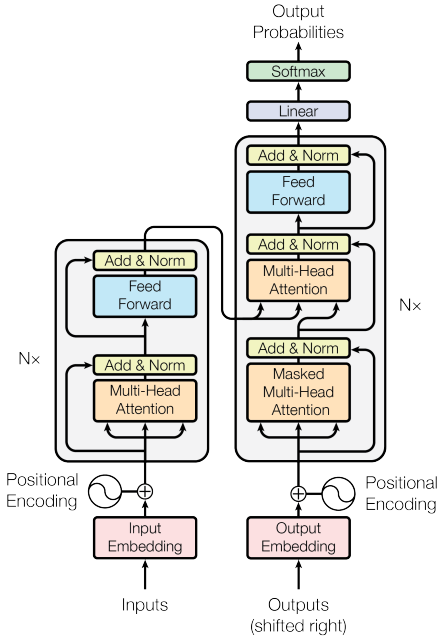
$$W \sim d_{model} \times K$$

Input and output embedding matrices can be shared or separate

Positional Encoding

Transformers do not maintain sequence position explicitly

- The encoder and decoder stacks are identical, for all positions
- Unlike a RNN encoder/decoder (bottom right), there is no left-to-right dependency



A symbol's position in a sequence is encoded in the symbol's representation

- "positional encodings" are added to the word embeddings prior to the input layer of the encoder and decoder layer
  - Without positional encoding, sequences are just bags-of-words
- The positional encoding at position *p* is a vector *PE(p)* of dimension d_{model} x 1

$$\mathrm{PE}(p)[k] = \begin{cases} \cos(p/10000^{(k-1)/d}) & k \text{ odd} \\ \sin(p/10000^{(k/d)}) & k \text{ even} \end{cases} \quad \text{for } k = 1 \ldots, d_{model}$$

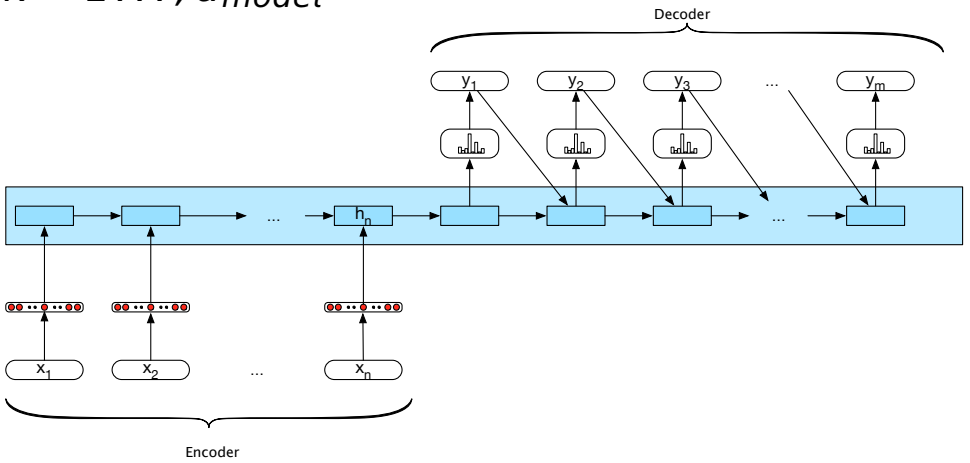
Example: encoding of *hat* in position 6 of a string:

PE(6) + W [hat]

positional encoding

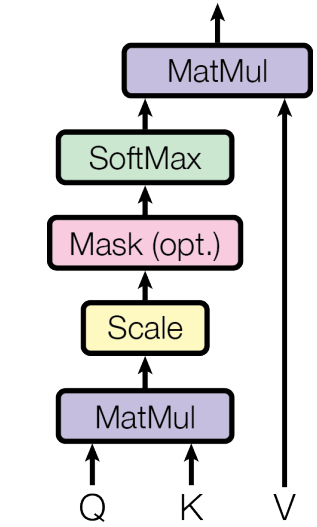
word embedding

one-hot

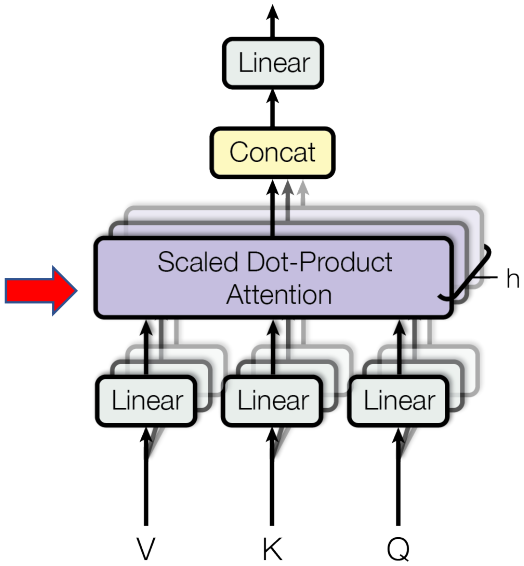


UNIVERSITY OF CAMBRIDGE

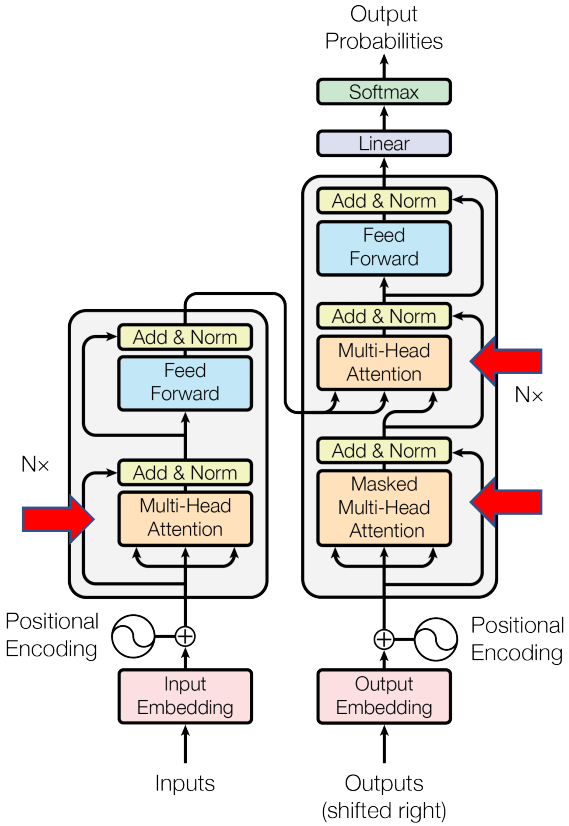
Multi-Head Attention



Scaled Dot-Product Attention



Multi-Head Attention



UNIVERSITY OF CAMBRIDGE

Aside: Retrieval with Queries, Keys, and Values

Suppose we have N key-value pairs $\{(k_n, v_n)\}_{n=1}^{N}$ where $k_n \sim 1 \times d_k$ and $v_n \sim 1 \times d_v$.

Arrange the keys and values into matrices K and V

$$K = \begin{pmatrix} k_1 \\ \vdots \\ k_N \end{pmatrix}_{N \times d_k} \qquad V = \begin{pmatrix} v_1 \\ \vdots \\ v_N \end{pmatrix}_{N \times d_v}$$

Let q be a $1 \times d_k$ dimensional query. qK^T is a $1 \times N$ vector whose n^{th} element is

$$(qK^T)_n = \sum_{i=1}^{d} q_i K^T_{i,n} = \sum_{i=1}^{d} q_i K_{n,i} = q \cdot k_n$$

Therefore (qK^T) contains the dot-products between the query and the keys.

Softmax Retrieval:

$$\text{softmax}(qK^T)\, V = \sum_{n=1}^{N} \text{softmax}([q \cdot k_i]_{i=1}^{N})_n \, v_n$$

- Interpolation of values, weighted by the softmax of the inner products between keys and the query

Aside: Batch Retrieval with Multiple Queries

Suppose we have M queries q_m, $m = 1, \ldots, M$, $\quad q_m \sim 1 \times d_k$.

Let K and V be as before, and $Q = \begin{pmatrix} q_1 \\ \vdots \\ q_M \end{pmatrix}_{M \times d_k}$

It follows that $(QK^T)_{i,j}$ is the dot-product between the i^{th} query and the j^{th} key:

$$QK^T = \begin{pmatrix} q_1 \cdot k_1 & q_1 \cdot k_2 & \cdots & q_1 \cdot k_N \\ q_2 \cdot k_1 & q_2 \cdot k_2 & \cdots & q_2 \cdot k_N \\ \vdots & & \vdots & \vdots \\ q_M \cdot k_1 & q_M \cdot k_2 & \cdots & q_M \cdot k_N \end{pmatrix}$$

Applying the softmax operation row-wise, the m^{th} row is the result from the query q_m

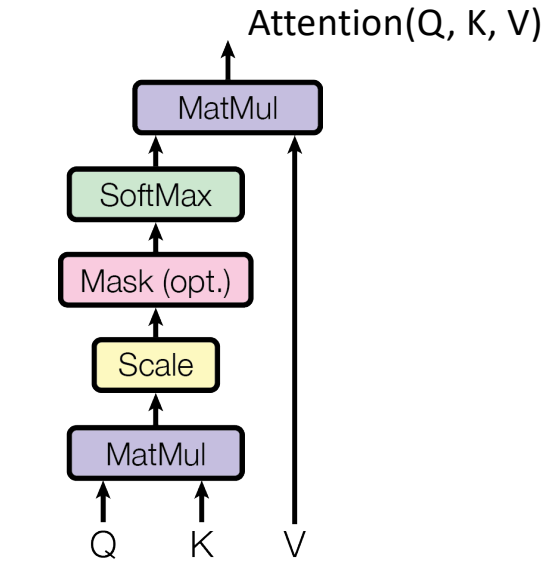
$$\text{softmax}(QK^T)\,V = \begin{pmatrix} \text{softmax}(q_1 \cdot k_1, q_1 \cdot k_2, \cdots, q_1 \cdot k_N) \\ \vdots \\ \text{softmax}(q_M \cdot k_1, q_M \cdot k_2, \cdots, q_M \cdot k_N) \end{pmatrix} V$$

Scaled Dot-Product Attention

Queries, Keys, and Values are packed into matrices *Q, K, V*

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

- Queries and Keys have dimension d_k
  - d_k is used as a fixed **scale** factor
- Values have dimension d_v



Scaled Dot-Product Attention

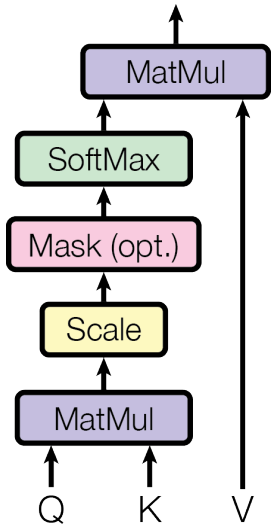
Self-Attention

- Suppose the input is a sequence $\mathbf{x} = x_1, \ldots, x_M$, $x_i \sim 1 \times d_{model}$
- A **Self-Attention** layer is parameterized by matrices (W^Q, W^K, W^V)
  - $W^Q, W^K \sim d_{model} \times d_k$ $\quad W^V \sim d_{model} \times d_v$ $\quad$ and $d_k = d_v$
  - These are learned during parameter estimation
- **Queries**, **Keys**, and **Values** are found as linear transforms of the input **x**

$$X = \begin{pmatrix} x_1 \\ \\ x_M \end{pmatrix} \qquad \begin{aligned} Q &= X\,W^Q \sim M \times d_k \\ K &= X\,W^K \sim M \times d_k \\ V &= X\,W^V \sim M \times d_v \end{aligned}$$

- for each x_i : $k_i = x_i W^K$, $q_i = x_i W^Q$, $v_i = x_i W^V$

Attention($X W^Q$, $X W^K$, $X W^V$)



$\text{softmax}(Q K^T) V$

Self-Attention as a Feed-Forward Network

Self-attention Transforms an input sequence $\mathbf{x} = x_1, \ldots, x_M$ into an output sequence $\mathbf{x'} = x'_1, \ldots, x'_M$

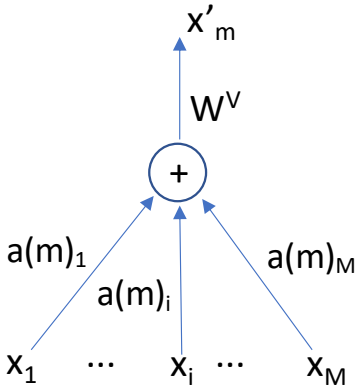
Each x_m can be produced as the output of a feed-forward network

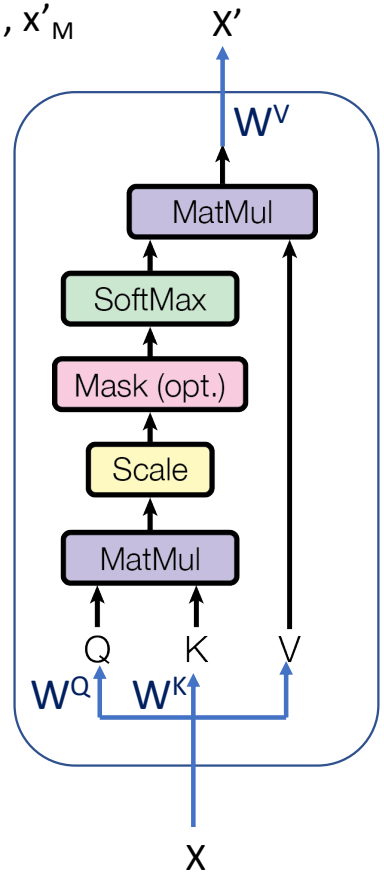
- For each x_i : $k_i = x_i W^K$, $q_i = x_i W^Q$
- Recall that the m^{th} row of QK^T is $[q_m \cdot k_1 \ldots q_m \cdot k_M]$

We can write the m^{th} row of softmax(QK^T) W^V as

$$a(m) = \text{softmax}([q_m \cdot k_1, \ldots, q_m \cdot k_M])$$

$$x'_m = [\sum_{i=1}^{M} a(m)_i \, x_i] \, W^V$$





a(m) is an attention vector

Masked self-attention

Enforces **causality** in self-attention - attention is not allowed to `peek` into the future
- x_m can only be a function of $x_1 \ldots x_{m-1}$

Recall that the QK^T matrix looks like

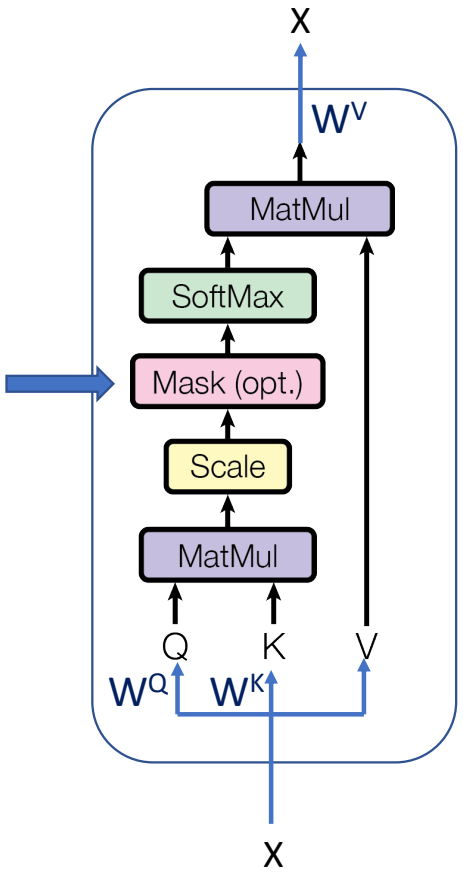
$$QK^T = \begin{pmatrix} q_1 \cdot k_1 & q_1 \cdot k_2 & \cdots & q_1 \cdot k_M \\ q_2 \cdot k_1 & q_2 \cdot k_2 & \cdots & q_2 \cdot k_M \\ \vdots & & \vdots & \vdots \\ q_M \cdot k_1 & q_M \cdot k_2 & \cdots & q_M \cdot k_M \end{pmatrix}$$

Masking sets the upper-triangle elements of QK^T to $-\infty$

$$\text{Mask}\, QK^T = \begin{pmatrix} q_1 \cdot k_1 & -\infty & -\infty & \cdots & -\infty \\ q_2 \cdot k_1 & q_2 \cdot k_2 & -\infty & \cdots & -\infty \\ \vdots & \vdots & & \vdots & -\infty \\ q_M \cdot k_1 & q_M \cdot k_2 & \cdots & q_M \cdot k_{M-1} & q_M \cdot k_M \end{pmatrix}$$

In the softmax operation, these masked values will be 0 :

$$\begin{aligned} y_m &= [\sum_{i=1}^{M} a(m)_i \, x_i \,] \, W^V \\ a(m) &= \text{softmax}([\, q_m \cdot k_1, \cdots, q_m \cdot k_m, -\infty \cdots - \infty \,]) \\ a(m)_i &= 0 \text{ for } i > m \end{aligned}$$



Multi-Head Attention

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

- Individual heads tend to focus on just one position

Use *h* attention heads:

- Scaled Dot-Product Attention, each with its own parameters

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \quad \text{for } i = 1, \ldots, h$$

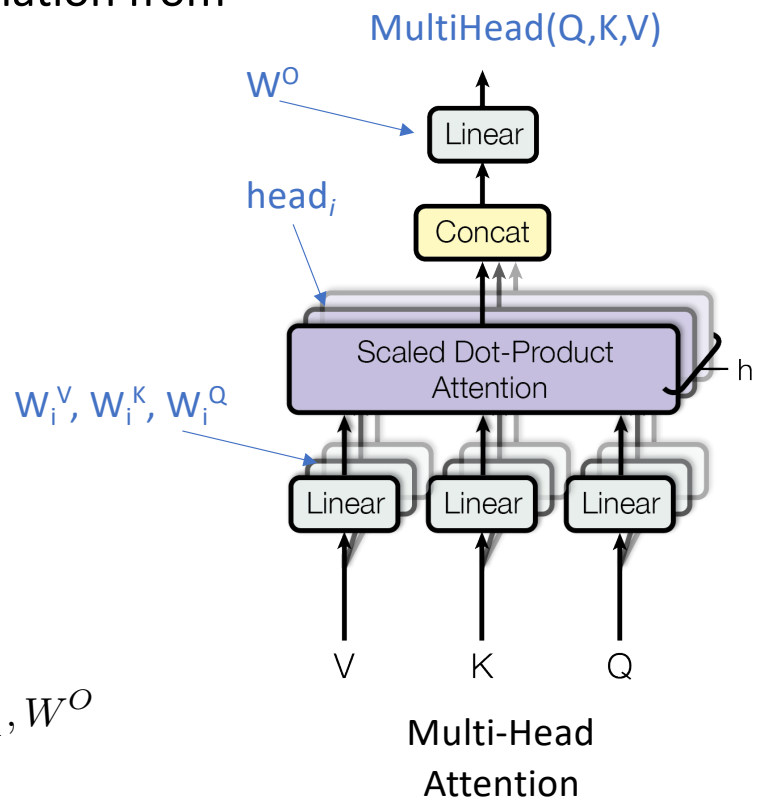
  - output of each head has d_v columns

Concatenate all *h* attention heads

$$\underbrace{\text{MultiHead}(Q, K, V)}_{d_{model} \text{ columns}} = \underbrace{\text{Concat}(\text{head}_1, \ldots, \text{head}_h)}_{hd_v \text{ columns}} W^O$$

Multi-Head Attention layer is parameterized by $\{W_i^Q, W_i^K, W_i^V\}_{i=1}^h, W^O$

- $W^O \sim h\ d_v \times d_{model}$



Multi-Head Attention

Residual connection with Layer Normaliation

Add & Norm

- Residual Learning

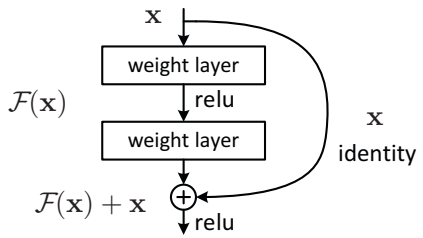
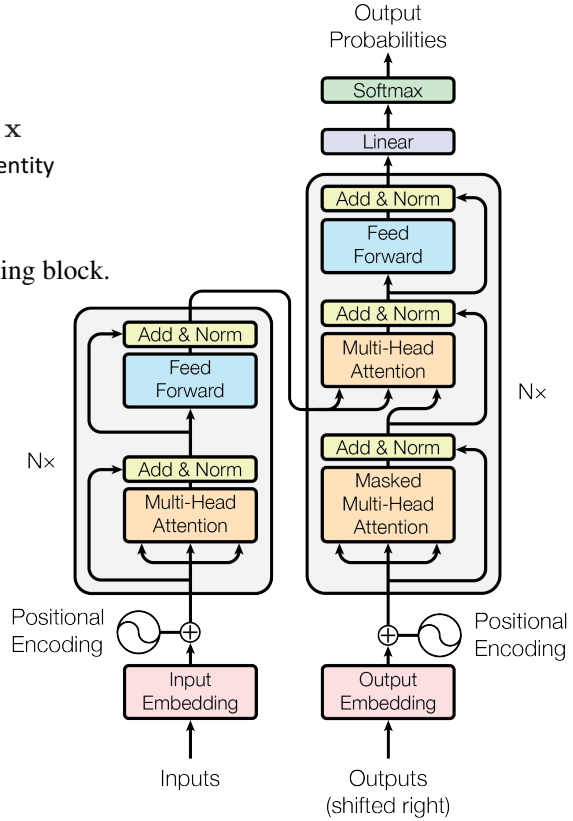


Figure 2. Residual learning: a building block.

Layer Normalisation

- Find the mean and variance across all units in a layer
- Normalise the input prior to the non-linearity
- Layer normalization performs exactly the same computation at training and test times



Position-wise Feed-Forward Networks

Each encoder and decoder layer contains a fully connected feed-forward network

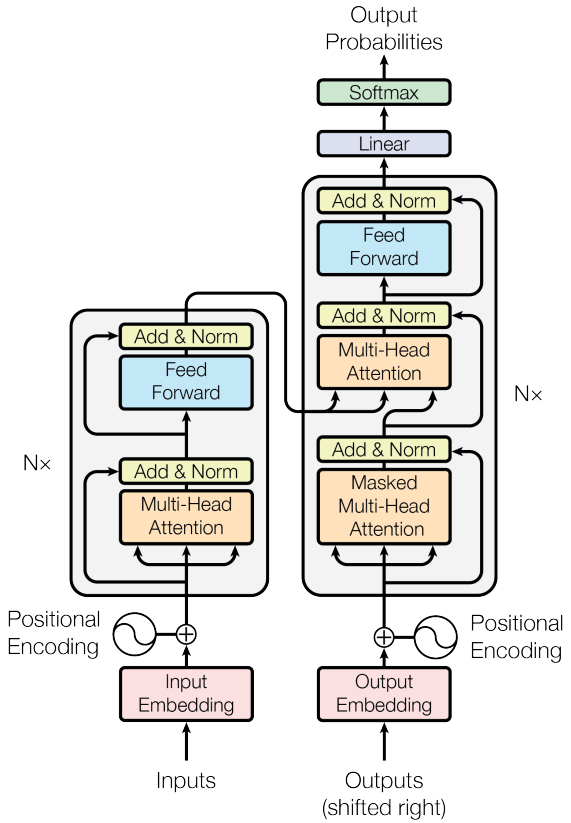
- applied to each position separately and identically

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

- FFN's also can be viewed as key-value memories*

$$\mathrm{FFN}(x) = f(x\ W_1)\ W_2$$

query key value

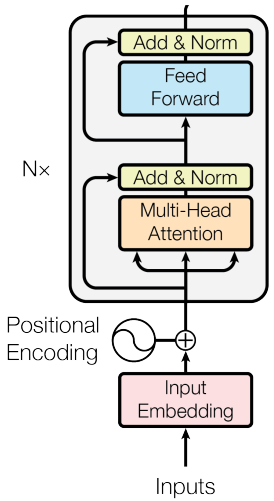


UNIVERSITY OF CAMBRIDGE

*Geva et al. Transformer Feed-Forward Layers Are Key-Value Memories arXiv:2012.14913v1

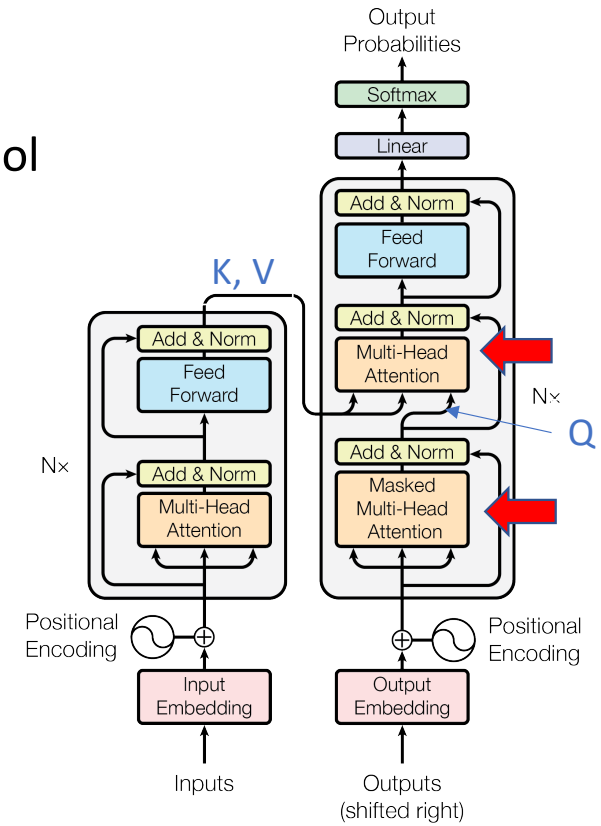
Encoder Summary

- Entire input sequence is processed together
- Input Embedding, with positional encodings
- N layers
- Each layer has
  - Multi-head Self-attention
  - Residual connections with layer normalisation
  - Position-wide Feed-Forward Networks, ReLU



Decoder Summary

- Entire output sequence can be processed at once, or symbol by symbol
  - due to causality in the masked self-attention
- Output embedding with positional encodings
- Softmax layer at output
- N identical layers
- Each layer has
  - Masked Multi-head attention
  - Encoder-Decoder attention
    - **Queries** come from the previous decoder layer
    - **Keys** and **Values** come from the output of the encoder
    - every position in the decoder can attend to every position in the input



Translation via Sequence-to-Sequence Models

Aim: define conditional distributions over target language sentences, given a source sentence

- Source sentence **x** = $x_1 \ldots x_n$
- Target sentence **y** = $y_1 \ldots y_m$,

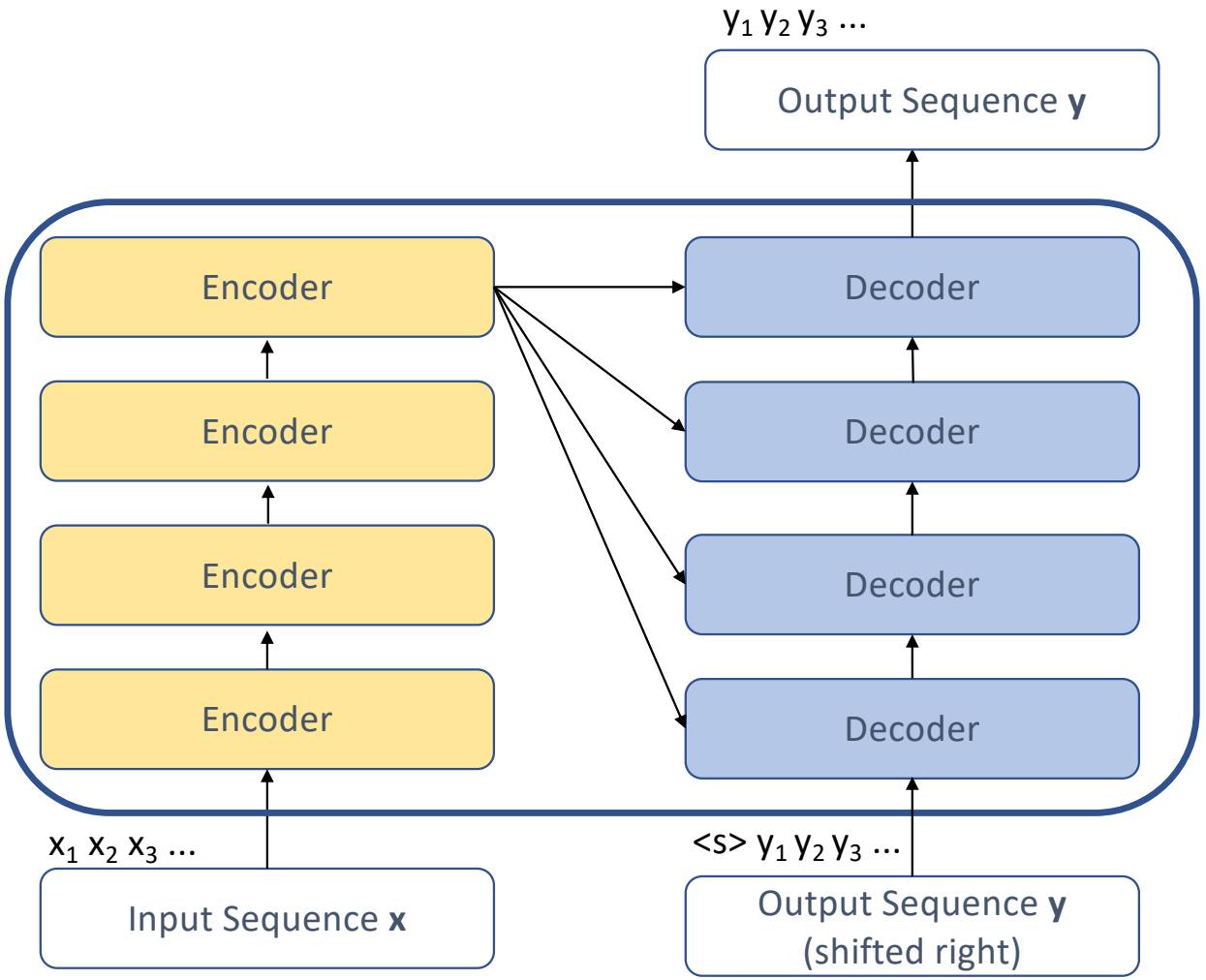
The Transformer defines a conditional probability over target language sequences: P(**y** | **x**; θ)

$$P(y|x;\theta) = \prod_{i=1}^{m} \underbrace{P(y_i|y_{<i}, x;\theta)}_{\text{Transformer}}$$

Training:

- parallel text: $\{x^n, y^n\}_{n=1}^{N}$
- use stochastic gradient descent to maximize $\prod_{n=1}^{N} P(y^n|x^n;\theta)$ with respect to θ

Encoder-Decoder Structure



Encoder and Decoder can be used separately

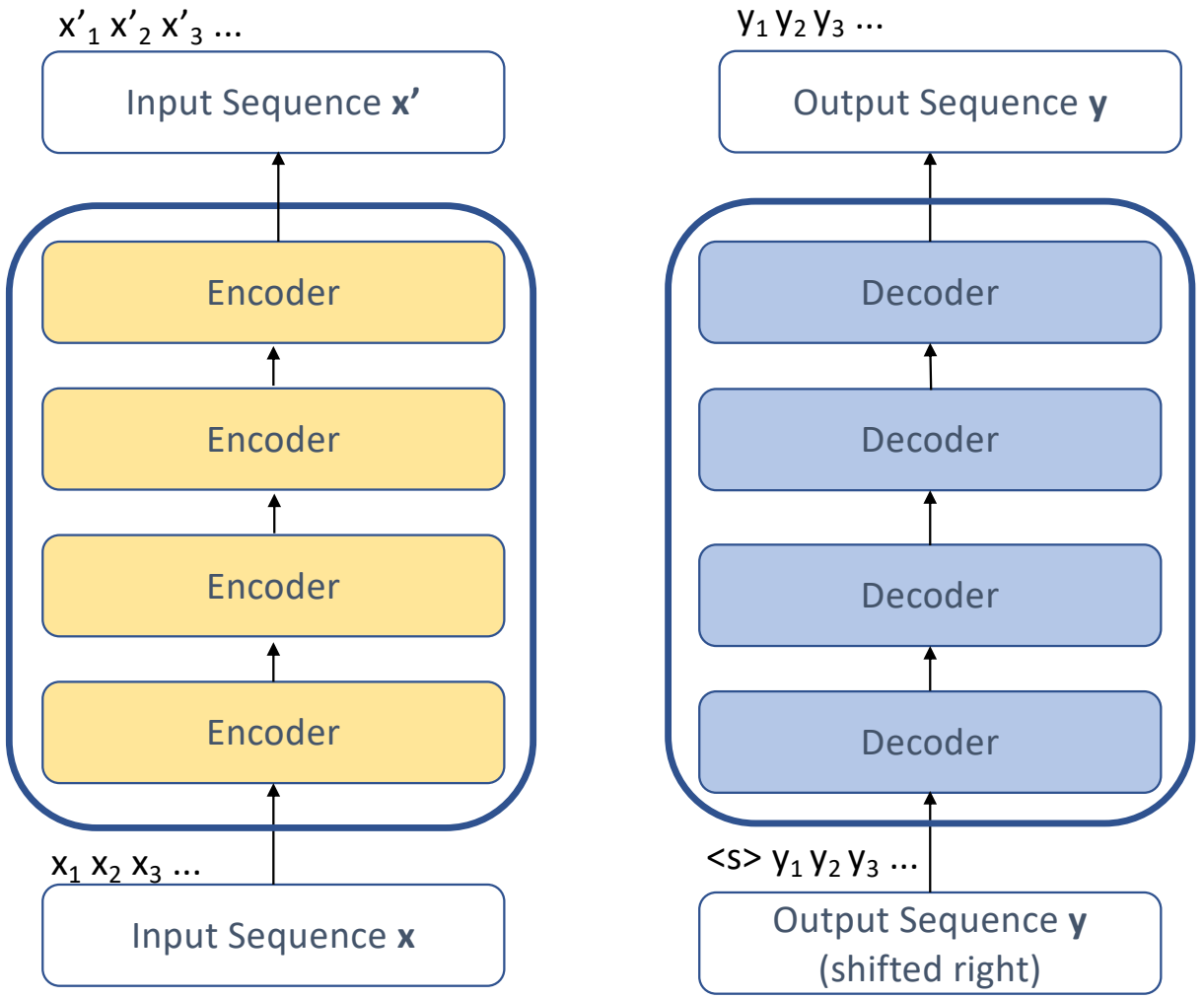
Recall that the decoder masking enforces *causality*

Encoder: **BERT**
- assigns probabilities at the sequence level

$$P(x'|x;\theta)$$

Decoder: **GPT-2**
- assigns predictive probabilities, like a language model

$$\prod_i P(y_i|y_{<i};\theta)$$



Summary

- Reviewed the Transformer architecture, as originally presented
- Links to memory networks and attention as `soft' query / key / value retrieval
- Very briefly introduced encoder-only and decoder-only networks
  - BERT
  - GPT-2

There is a vast literature on extensions and analyses to the Transformer

- focused on improving speed, memory use, ...*

* e.g. Tay et al. *Efficient Transformers: A Survey* , 2020, https://arxiv.org/abs/2009.06732