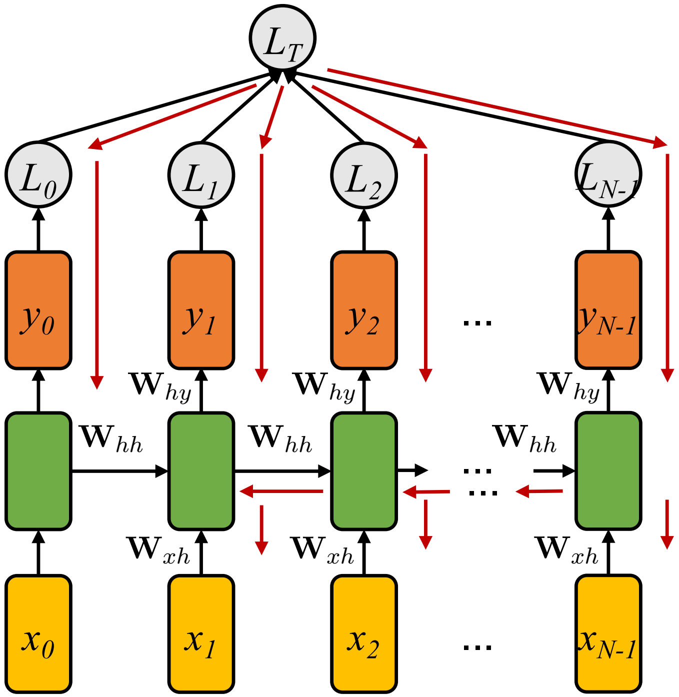
Machine Learning in Biomedical Sciences and Bioengineering

Lecture 9
Transformers

2025 version 1.00

James Choi

RNN architecture

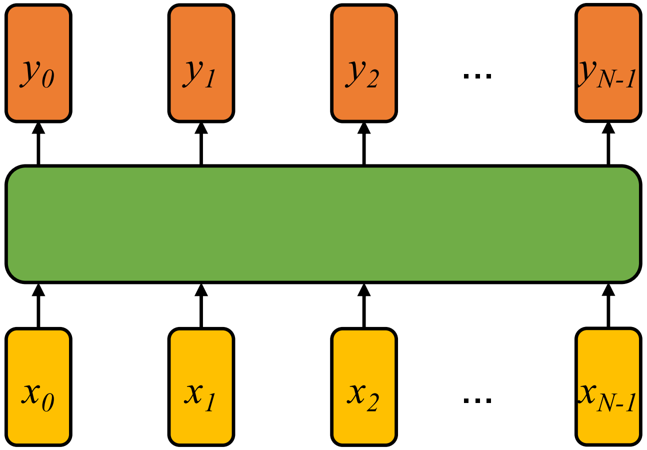


$$h_i = tanh(\mathbf{W}_{hh}^T h_{i-1} + \mathbf{W}_{xh}^T x_i)$$

$$\hat{y}_i = \mathbf{W}_{hy}^T h_i$$

- **Limitations with RNNs**
  - Poor long-term memory
  - Encoding bottleneck
  - Slow compute (no parallelisation)

RNN architecture



- **Potential solution (?):**
  - Create a single block of weights?

- **Many problems...**
  - NO order, no temporal dependence
  - NO long-term memory
  - NOT scalable

Part 1. Transformers

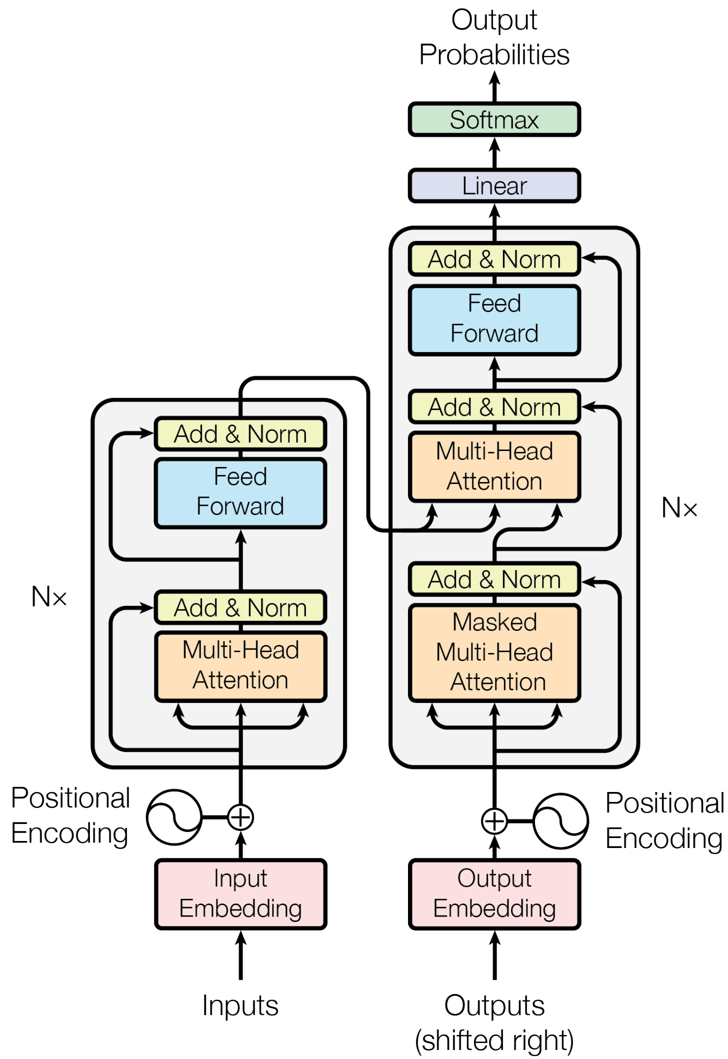
Part 2. Live Coding Demonstration

The Transformer

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.



Inputs

Input to the Transformer model:

tensor $\mathbb{R}^B \times \mathbb{R}^N$

B: batch size

N: sequence length

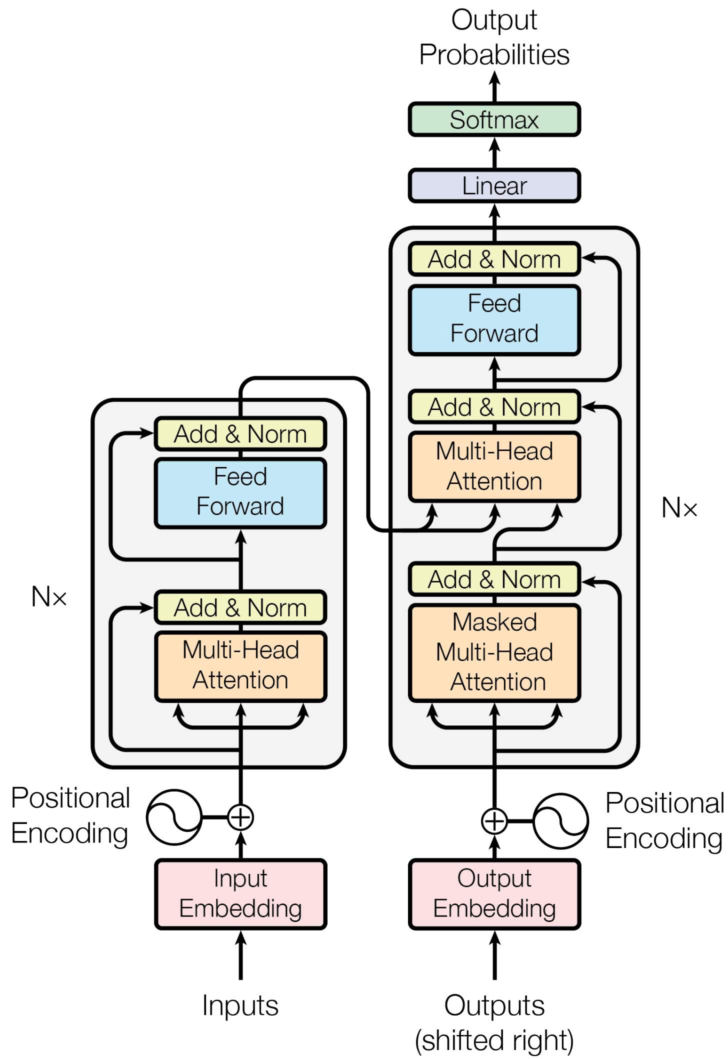
$\mathbb{R}^n = \{(x_1 \cdots x_n) : x_j \in \mathbb{R} \text{ for } j = 1, \cdots, n\}$

Input passes through an embedding layer that converts each one-hot token representation into a d_{model} dimensional embedding

tensor $\mathbb{R}^B \times \mathbb{R}^N \times \mathbb{R}^{d_{model}}$

The new tensor is then additively composed with positional encodings...

Passes through a multi-headed self-attention module.



Positional encoding

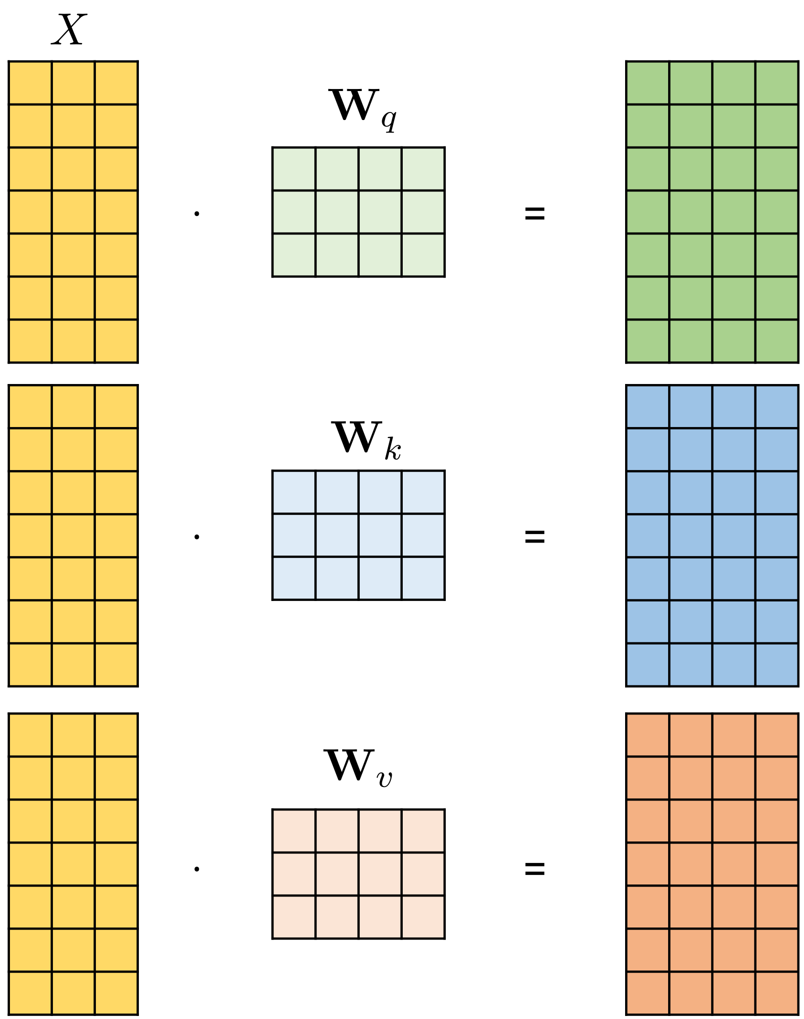
- **Goal:** inject the position information of each token into the input data.

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

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

pos: position
i: the dimension
d_{model}: model's dimension

Self-attention



Query: information that is being looked for

$$Q_h = X\mathbf{W}_q$$

Key: the context or reference

$$K_h = X\mathbf{W}_k$$

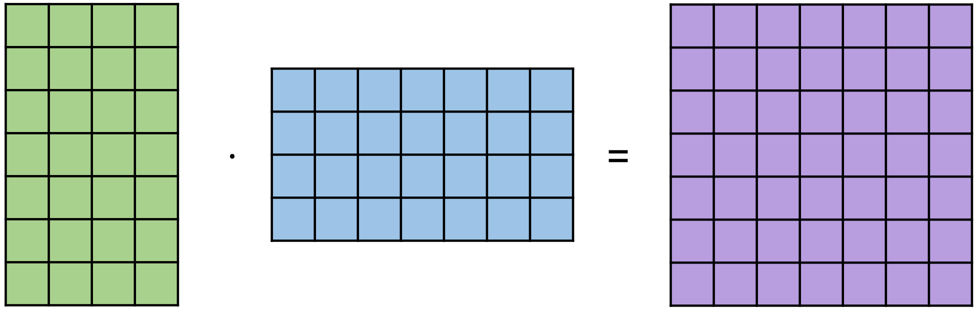
Value: content that is being searched for.

$$V_h = X\mathbf{W}_v$$

Self-attention

- How similar is the **Query** to the **Key**?
- Compute the following to calculate the pairwise similarity between each query and key:

$$\frac{QK^T}{\sqrt{d_k}} = \sqrt{d_k} \quad \cdot \quad =$$



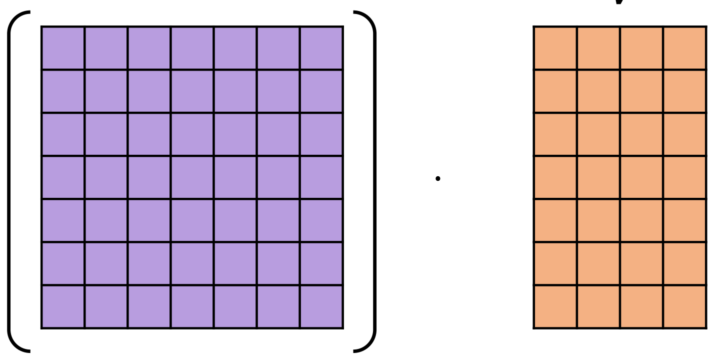
d_k: dimension of the queries and keys

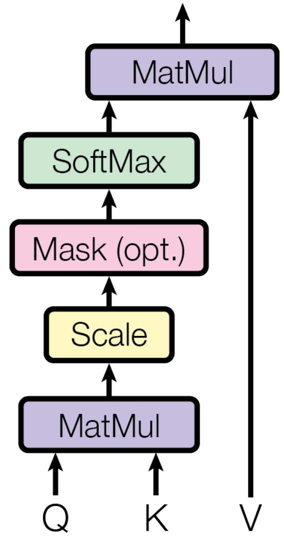
Self-attention

- Compute attention weighting

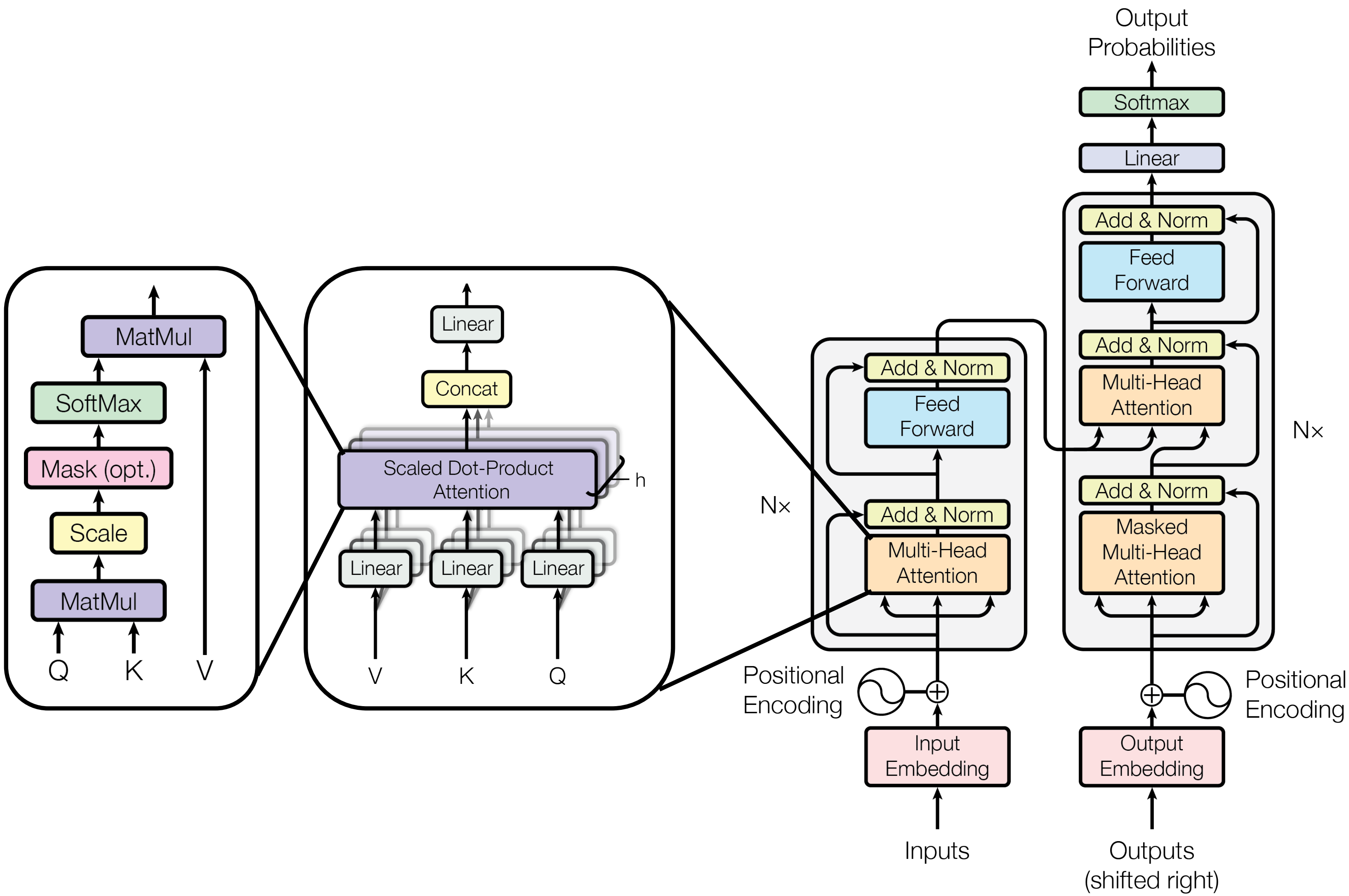
$\frac{QK^T}{\sqrt{d_k}}$ $\quad V$

$\mathrm{Attention}(Q, K, V) =$ softmax





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


MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Part 1. Transformers

Part 2. Live Coding Demonstration

Live coding demonstration

- Code adapted from...
  – https://github.com/pytorch/tutorials/blob/main/beginner_source/translation_transformer.py
  – This is a deprecated link. The live coding session code will be uploaded onto the online platform.