MACHINE LEARNING IN PHYSICS
TRANSFORMERS

HARRISON B. PROSPER

PHY6938

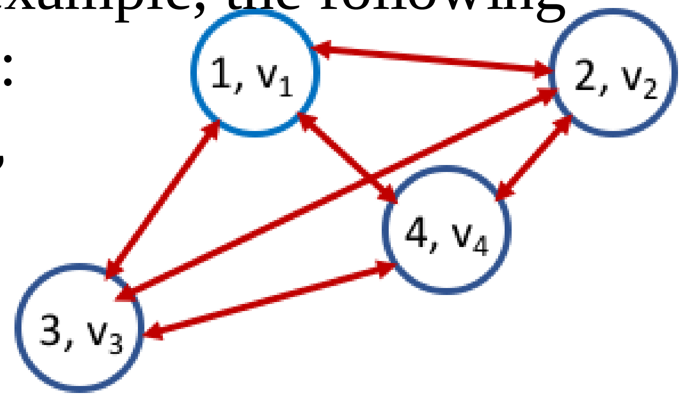
Recap: Graph Neural Networks

A GNN can be used to process data represented as a cloud of n points that reside in a d-dimensional vector space. The key components are:

1. An adjacency matrix, $\boldsymbol{A}$, of shape (n, n) that encodes edge information.
2. A matrix, $\boldsymbol{X}$, of shape (n, d) that encodes vertex information.
3. A graph convolution operation. For example, the following is used by the IceCube Collaboration:

$$\boldsymbol{Y} = [\boldsymbol{AX}, \boldsymbol{X}]w + bI,$$
$$\boldsymbol{X} = [\text{ReLU}(\boldsymbol{Y}), \boldsymbol{Y}]$$

where w and b are parameters.



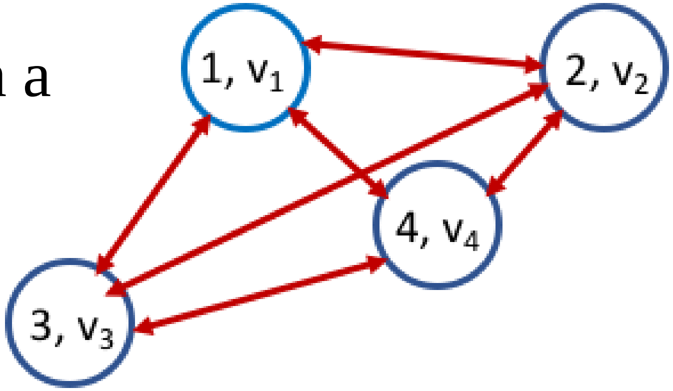
Recap: Graph Neural Networks

In the GNN tutorial (Lab08), given the input graph $G = (V, E)$ where $V = X$ are the n vertices and $E = A$ model the edges, we implemented the following graph convolution operation

$$X = \mathrm{ReLU}(XW^T + \mathrm{diag}B)$$
$$X = \mathrm{ReLU}(AXw + bI)$$

where W, B, w, and b are parameters.

The first operation embeds the vertices in a higher-dimensional space and the second operation is the graph convolution.



Introduction

1. Convolutional neural networks (CNN)
2. Autoencoder (AE)
3. Physics-informed neural networks (PINN)
4. Flow and diffusion models
5. Graph neural networks (GNN)
6. Transformer neural networks (TNN)

Sequence to Sequence Models

A sequence to sequence (seq2seq) model can be used to map one *sequence* of tokens to another. A word, part of a word, or a symbol are examples of tokens.

Examples:

1. the white house $\Longrightarrow$ la maison blanche
2. $\cosh^3(ax) + \tanh(bx) \Longrightarrow 1 + bx + \frac{3a^2x^2}{2} - \frac{b^3x^3}{3} + O(x^4)$

Tokens: the, white, house, la, maison, cosh, tanh, x, a, b, … etc.

Sequence to Sequence Models

Consider what a model needs to do to solve the following translation task:

the white house $\Longrightarrow$ la maison blanche

1. The model needs a vocabulary of tokens for each language, that is, words or parts of words: “the”, “la”, etc.

2. The model must pay *attention* to the order of “house” and “white” and of “blanche” and “maison”.

3. It must also pay *attention* to the association between “white” and “blanche” and “house” and “maison”.

Seq2Seq Models: Steps

A seq2seq model generally implements the following steps.

1. Tokenization: splitting a sequence into tokens.
2. Coding: assigning a unique integer to each unique token.
3. Embedding: representing each token by a vector.
4. Analysis: analyzing the input (source) sequence.
5. Synthesis: constructing the output (target) sequence.

Seq2Seq: Tokenization

The sequence $x = \tan(hx) + \sinh(gx)$ can be split into the tokens tan, sinh, (,), +, h, g, and x.

The set of unique tokens is called a vocabulary.

Example:

$$\tan(hx) + \sinh(gx)$$

$$x(g+h) + x^3\left(\frac{g^3}{6} + \frac{h^3}{3}\right) + x^5\left(\frac{g^5}{120} + \frac{2h^5}{15}\right) + O\left(x^6\right)$$

```
source vocabulary
{'<pad>': 0, '<sos>': 1, '<eos>': 2, '(': 3, ')': 4, '*': 5, '**': 6, '+': 7, '-': 8, '/': 9, '0'
: 10, '1': 11, '2': 12, '3': 13, '4': 14, '5': 15, '6': 16, '7': 17, '8': 18, '9': 19, 'a': 20, '
b': 21, 'c': 22, 'cos': 23, 'cosh': 24, 'd': 25, 'exp': 26, 'f': 27, 'g': 28, 'h': 29, 'm': 30, '
n': 31, 'sin': 32, 'sinh': 33, 'tan': 34, 'tanh': 35, 'x': 36}

target vocabulary
{'<pad>': 0, '<sos>': 1, '<eos>': 2, '(': 3, ')': 4, '*': 5, '**': 6, '+': 7, '-': 8, '/': 9, '0'
: 10, '1': 11, '2': 12, '3': 13, '4': 14, '5': 15, '6': 16, '7': 17, '8': 18, '9': 19, 'O(x**6)':
20, 'a': 21, 'b': 22, 'c': 23, 'd': 24, 'f': 25, 'g': 26, 'h': 27, 'm': 28, 'x': 29}
```

Seq2Seq: Coding/Padding

Since machines work with numbers it is necessary to map every token in a sequence to an integer.

Example: $x = \tan(hx) + \sinh(gx)$ is mapped to

$$x = 34, 3, 29,36,4, 7,33,3,28,36,4$$

$$\tan\left(hx\right) + \sinh\left(gx\right)$$

$$x\left(g + h\right) + x^{3}\left(\frac{g^{3}}{6} + \frac{h^{3}}{3}\right) + x^{5}\left(\frac{g^{5}}{120} + \frac{2h^{5}}{15}\right) + O\left(x^{6}\right)$$

```
source vocabulary
{'<pad>': 0, '<sos>': 1, '<eos>': 2, '(': 3, ')': 4, '*': 5, '**': 6, '+': 7, '-': 8, '/': 9, '0'
: 10, '1': 11, '2': 12, '3': 13, '4': 14, '5': 15, '6': 16, '7': 17, '8': 18, '9': 19, 'a': 20, '
b': 21, 'c': 22, 'cos': 23, 'cosh': 24, 'd': 25, 'exp': 26, 'f': 27, 'g': 28, 'h': 29, 'm': 30, '
n': 31, 'sin': 32, 'sinh': 33, 'tan': 34, 'tanh': 35, 'x': 36}

target vocabulary
{'<pad>': 0, '<sos>': 1, '<eos>': 2, '(': 3, ')': 4, '*': 5, '**': 6, '+': 7, '-': 8, '/': 9, '0'
: 10, '1': 11, '2': 12, '3': 13, '4': 14, '5': 15, '6': 16, '7': 17, '8': 18, '9': 19, 'O(x**6)':
20, 'a': 21, 'b': 22, 'c': 23, 'd': 24, 'f': 25, 'g': 26, 'h': 27, 'm': 28, 'x': 29}
```

Seq2Seq: Coding/Padding

The sequence length of $x = 34, 3, 29, 36, 4, 7, 33, 3, 28, 36, 4$ is 11 tokens. But for many models, sequences must be of *equal* length, which is done by ***padding***, and they must be ***delimited***:

$$x = 1, 34, 3, 29, 36, 4, 7, 33, 3, 28, 36, 4, 0, 0, 0, 2$$

$\tan(hx) + \sinh(gx)$

$$x(g+h) + x^3\left(\frac{g^3}{6} + \frac{h^3}{3}\right) + x^5\left(\frac{g^5}{120} + \frac{2h^5}{15}\right) + O\left(x^6\right)$$

```
source vocabulary
{'<pad>': 0, '<sos>': 1, '<eos>': 2, '(': 3, ')': 4, '*': 5, '**': 6, '+': 7, '-': 8, '/': 9, '0'
: 10, '1': 11, '2': 12, '3': 13, '4': 14, '5': 15, '6': 16, '7': 17, '8': 18, '9': 19, 'a': 20, '
b': 21, 'c': 22, 'cos': 23, 'cosh': 24, 'd': 25, 'exp': 26, 'f': 27, 'g': 28, 'h': 29, 'm': 30, '
n': 31, 'sin': 32, 'sinh': 33, 'tan': 34, 'tanh': 35, 'x': 36}

target vocabulary
{'<pad>': 0, '<sos>': 1, '<eos>': 2, '(': 3, ')': 4, '*': 5, '**': 6, '+': 7, '-': 8, '/': 9, '0'
: 10, '1': 11, '2': 12, '3': 13, '4': 14, '5': 15, '6': 16, '7': 17, '8': 18, '9': 19, 'O(x**6)':
20, 'a': 21, 'b': 22, 'c': 23, 'd': 24, 'f': 25, 'g': 26, 'h': 27, 'm': 28, 'x': 29}
```

Seq2Seq: Embedding

An embedding space, a key idea in many machine learning models, is a vector space in which data elements, here tokens, are represented as vectors in that space.

Example 1: In our GNN implementation (Lab08), each point (p_T, η, ϕ) is mapped to a vector in a 40-dimensional vector space, where the mapping is determined during training.

Example 2: In our seq2seq tutorial (Lab09), each token and its ordinal position is mapped to a 64-dimensional vector space.

Seq2Seq: Analysis/Synthesis

- Seq2seq models, like recurrent neural networks (RNN), long short-term memories (LSTM), and transformer neural networks (TNN) – the subject of today's lecture – analyze and synthesize sequences in different ways.

- Moreover, some classes of seq2seq models (e.g., RNN, LSTM) analyze sequences one token at a time.

- Transformers were a breakthrough, in part, because tokens are analyzed in *parallel*.

TRANSFORMER
NEURAL NETWORKS

Attention Is All You Need

In 2017, in a seminal paper: https://arxiv.org/abs/1706.03762
Google researchers introduced a transformer neural network (TNN) with a highly successful analysis and synthesis method.

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

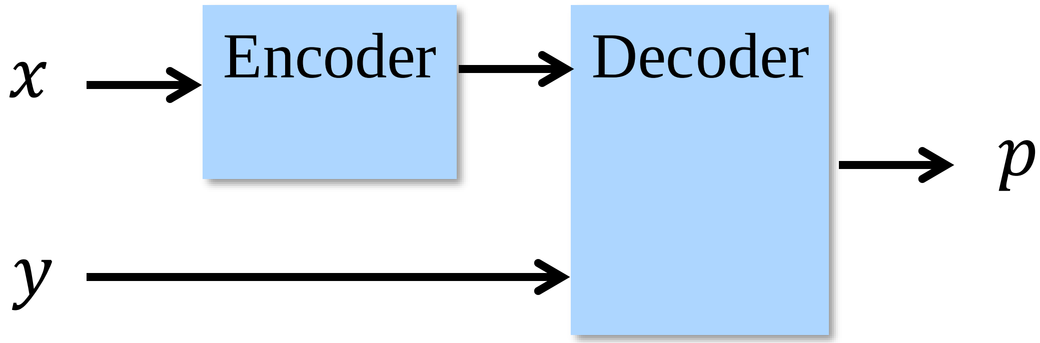
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

The Transformer

- Transformers are encoder-decoder models that are trained using supervised learning with cross-entropy loss.

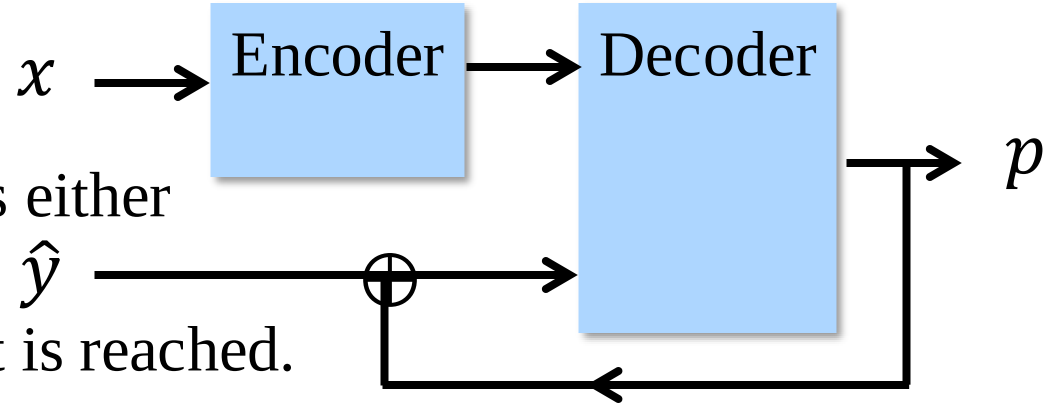


- During training, the source sequence x is analyzed by the encoder and the analyzed sequence together with the target sequence y is sent to the decoder, which analyzes the latter.
- The decoder then predicts the probabilities p for the next token for each of the *sub-sequences* of y and the mean cross-entropy loss computed from these probabilities is minimized.

The Transformer

After training, the model is used autoregressively.

- The source sequence x is analyzed by the encoder and the *predicted* sequence $\hat{y}$ is initialized to the start-of-sequence token (<sos>).
- The analyzed sequence x together with $\hat{y}$ are passed to the decoder, which predicts probabilities for the next token for each sub-sequence of $\hat{y}$. These probabilities are used to select the next token, which is appended to $\hat{y}$. The cycle repeats until the next token is either the end-of-sequence or a token count limit is reached.



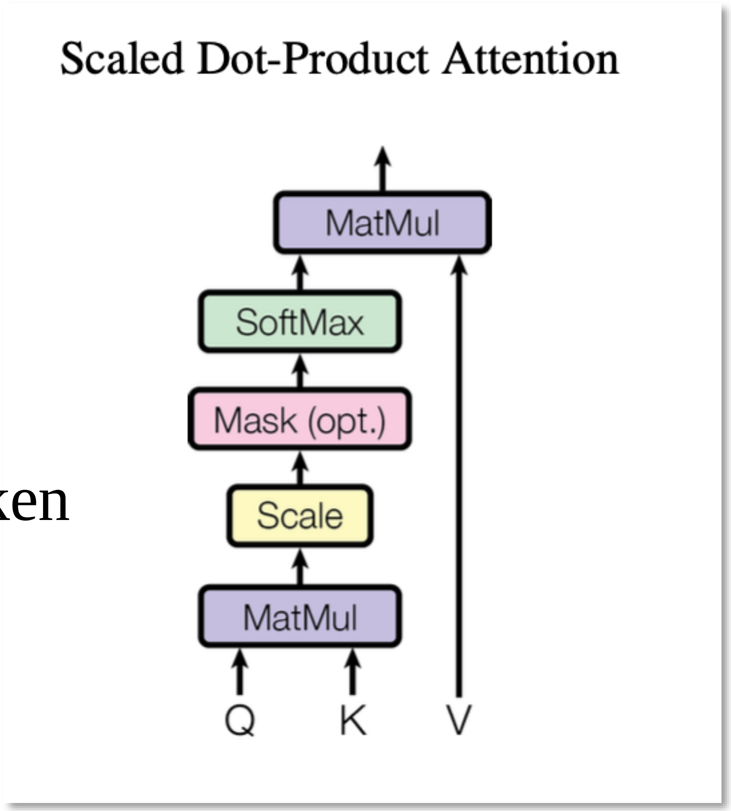
Attention

At the heart of Google's breakthrough transformer model is the following expression

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

which captures some notion of a token "paying attention to" other tokens.

Q: query, K: key, V: value



https://arxiv.org/abs/1706.03762

Attention

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

Source *or* target self attention:

Q, K, and V are the same matrix of tokens (represented by vectors in the embedding space)

Source *to* target attention:

Q models the target tokens.

K and V model the source tokens.

Attention: Token Vectors

But in what sense does

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

capture the notion of a token "*paying attention to*" other tokens?

Consider a sequence of 2 tokens, each represented by a vector.

The matrices can be viewed as a columns of vectors, one for each token:

$$Q = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}, K = \begin{bmatrix} k_1 \\ k_2 \end{bmatrix}, V = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

Attention: Dot Products

The matrix (outer) product QK^T is given by

$$QK^T = \begin{bmatrix} q_1 \cdot k_1 & q_1 \cdot k_2 \\ q_2 \cdot k_1 & q_2 \cdot k_2 \end{bmatrix}$$

Self attention is a matrix of dot products between source tokens or between target tokens.

Attention is a matrix of dot products between source and target tokens.

The key idea is that the dot product measures the *degree of association* between any pair of tokens.

Attention: Scaled Dot Products

The matrix of *scaled* dot products, after `softmax`, is given by

$$\text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right) = \begin{bmatrix} \exp\left(\frac{q_1 \cdot k_1}{\sqrt{d}}\right)/a_1 & \exp\left(\frac{q_1 \cdot k_2}{\sqrt{d}}\right)/a_1 \\ \exp\left(\frac{q_2 \cdot k_1}{\sqrt{d}}\right)/a_2 & \exp\left(\frac{q_2 \cdot k_2}{\sqrt{d}}\right)/a_2 \end{bmatrix}$$

where a_1 and a_2 are normalization factors that ensure each row sums to one.

The above matrix of *normalized* weights is then multiplied by the matrix V. The upshot is that each token is associated with a weighted sum of token vectors, presumably, the ones to which a given token "*pays attention*".

Transformer: Some Details

The transformer is a rather complicated function as is evident from this figure from the 2017 Google paper.

But let's work through it step by step. (Details can be found in Lab09)

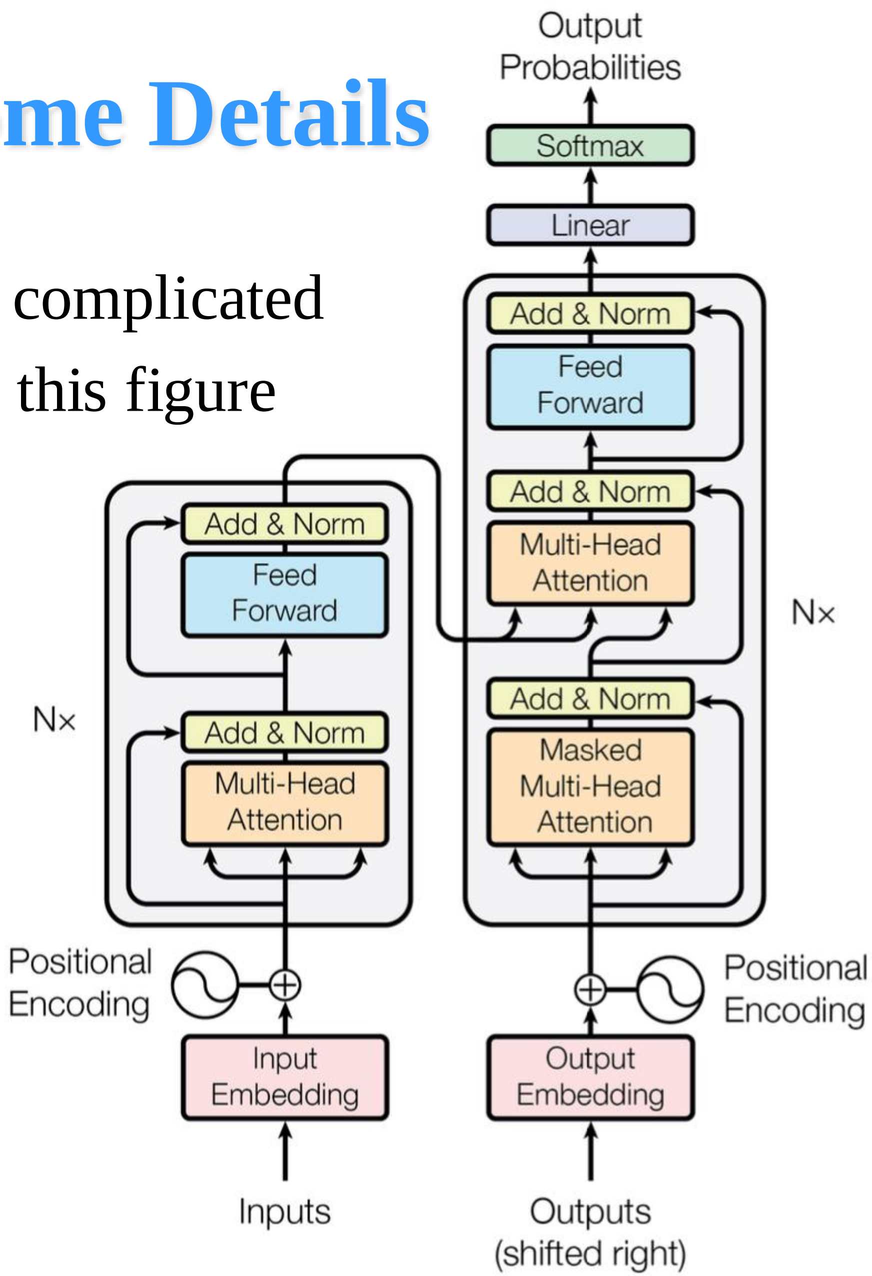
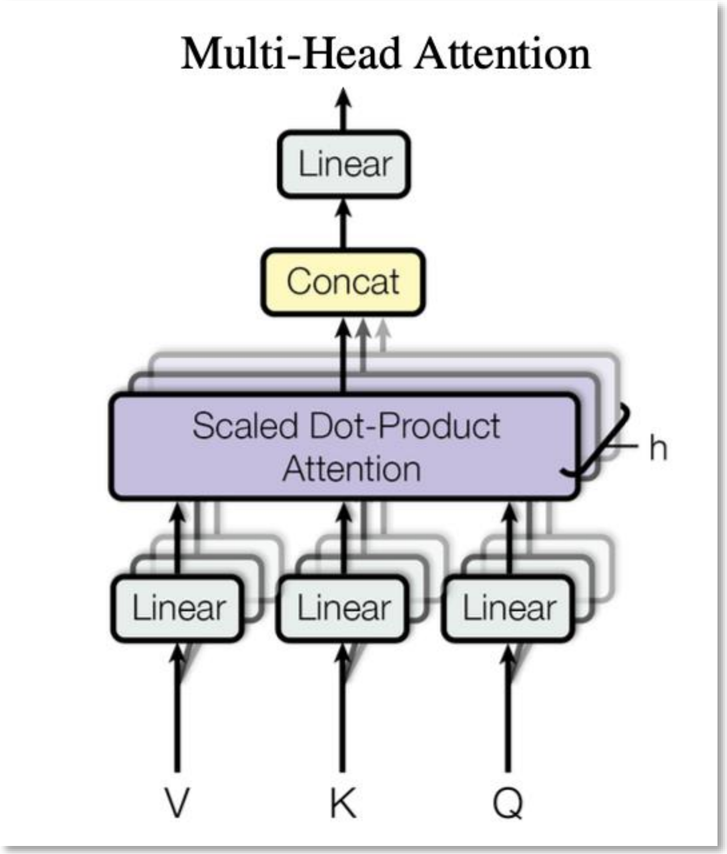


Figure 1: The Transformer - model architecture.

Transformer: **Some Details**

The key component of this function is the so-called multi-head attention.



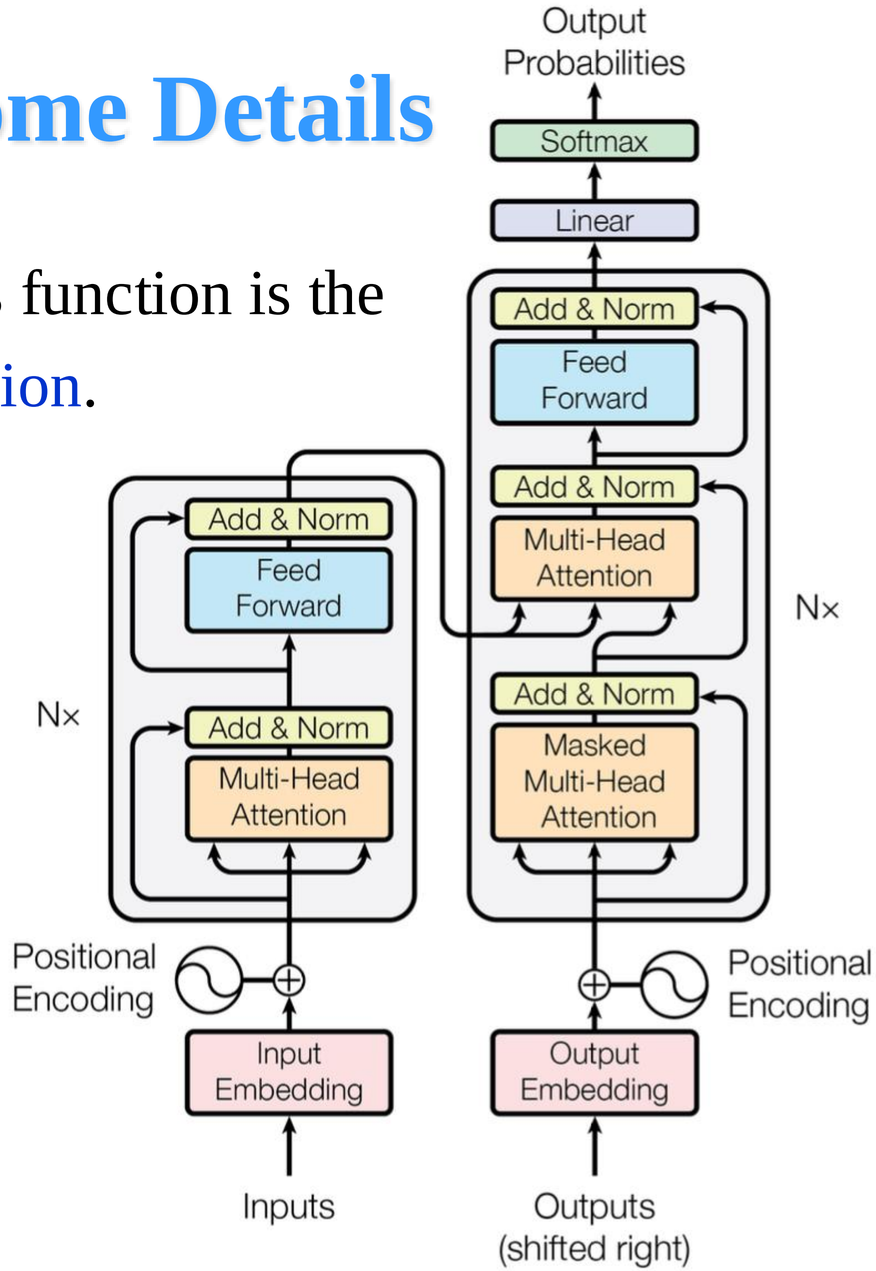
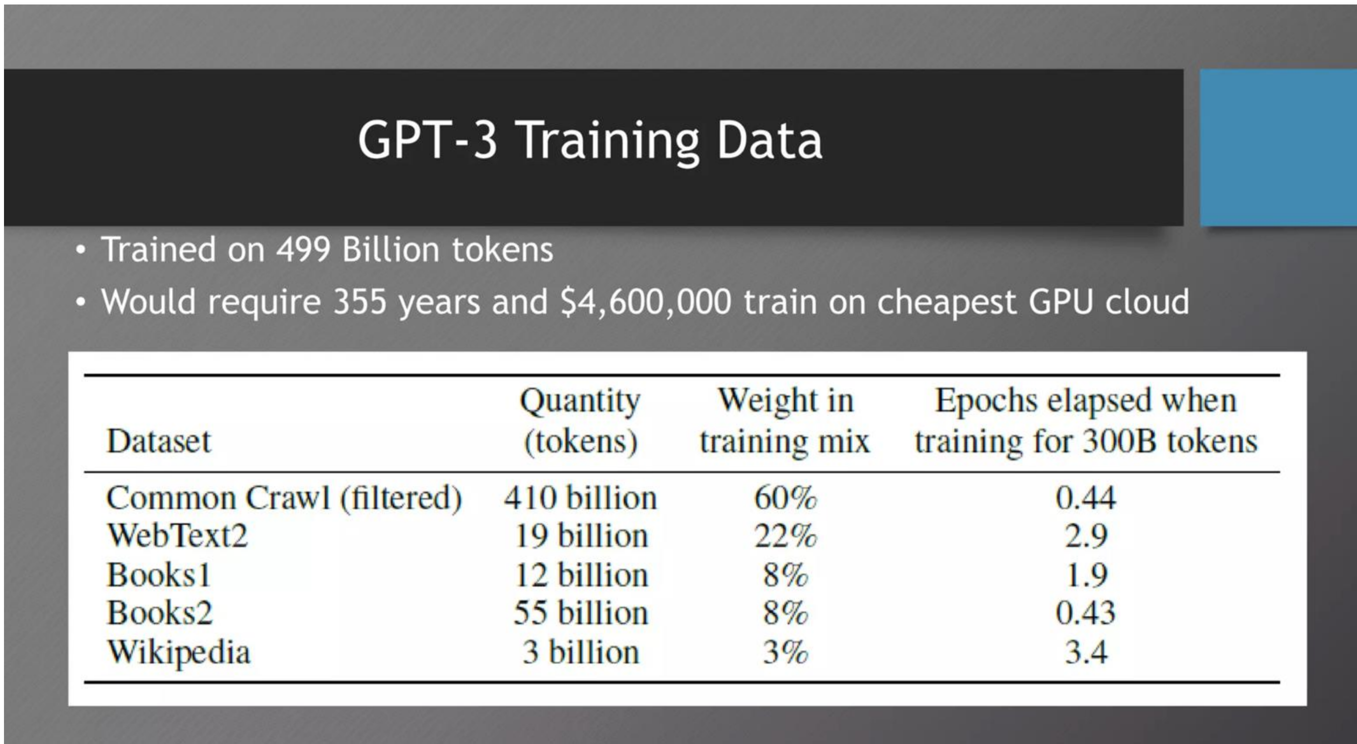


Figure 1: The Transformer - model architecture.

Transformer: ChatGPT

The GPT version 3 Training Data



Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Credit: Steve Omohundro / OpenAI

https://steveomohundro.com/

https://chat.openai.com/c/6ac10f83-6693-43c9-9825-8f143f7006bc

Transformer: ChatGPT

The GPT version 3 model that underlies the chatbot ChatGPT is a function with 175 billion parameters!

- Context window of 2,048 tokens
- 96 transformer layers
- 96 self-attention heads, each 128 dimensional
- 12,288 units in bottleneck layer, 49,152 in feed forward layer
- Batch size of 3.2M samples
- Learning rate .6*10^-4

Credit: Steve Omohundro / OpenAI

https://steveomohundro.com/

https://chat.openai.com/c/6ac10f83-6693-43c9-9825-8f143f7006bc

Summary

- The transformer neural network has revolutionized large-language models (LLM), such as the one that underpins the chatbot ChatGPT.

- Advantages
  - All tokens processed simultaneously during training.
  - Attention makes it possible to handle much longer sequences than was possible in previous models.
- Disadvantages
  - Complexity
  - Training time