Introduction & Transformer

Pretrained Models 2024 S | MSc CogSys

Meng Li

meng.li@uni-potsdam.de

Content

- Organization
  - Logistics
  - Course structure
  - Grading
  - Topics
- Transformer
  - Explain building blocks

Logistics

- Instructor: Meng Li
- Time: Tuesdays, 2:15-3:45pm (first meeting on April 16)
- Room: 2.14.0.32
- Course Management System: Moodle (for posting questions, uploading paper proposal / term paper)
- Office Hours: appointment-based

Logistics

- Course webpage: https://limengnlp.github.io/teaching/pretrain_24s/
  - We will maintain website for schedule, slides etc. here, not Moodle!

Date	Topic	Readings	Related Materials	Presenter	Slides
	Part 1: Introduction				
2024/04/16	Transformer	Attention is All you Need	The Annotated Transformer; The Illustrated Transformer; Huggingface NLP course on transformer	Meng	
2024/04/23	Pretrained Models with Encoder-only Architecture	(1) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding; (2) RoBERTa: A Robustly Optimized BERT Pretraining Approach; (3) ALBERT: A Lite BERT for Self-supervised Learning of Language Representations; (4) SpanBERT: Improving Pre-training by Representing and Predicting Spans; (5) XLNet: Generalized Autoregressive Pretraining for Language Understanding	A Primer in BERTology: What We Know About How BERT Works	Meng	

Course Structure

- This seminar is the first part of “pretrained models” course and focuses on transformer-based pretrained models with **encoder-only architecture**. It is a transition course
- Prerequisites
  - Essential understanding of neural networks;
  - Familiar with NLP tasks;
  - Intellectual curiosity

Course Structure

- We will focus on one topic each week.
  - The first two units: tutorials on transformer and pretrained models.
  - From the fourth week, there are two readings for each topic and two students will present in each unit. Each student will present one paper and lead followed discussion or activities.
  - The presentation should last **20-30 minutes** and leave **15-25 minutes for discussion or activities**.
  - Students are expected to read both papers every week, and **submit one question for each paper by Monday evening (23:59)**.

Course Structure

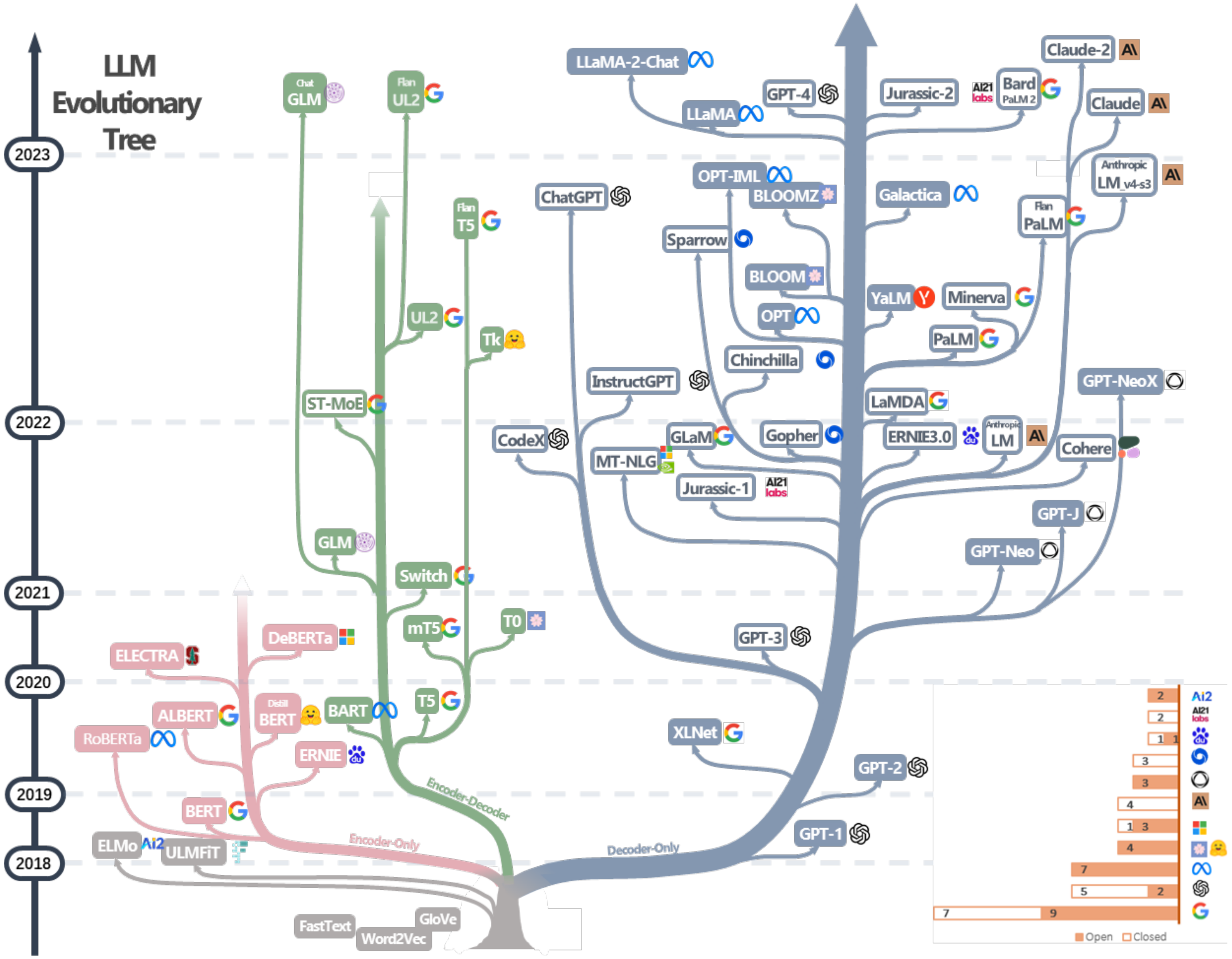
- **Registration**. If you would like to participate, you should directly register through PULS. In addition, please drop an email to meng.li (at) uni-potsdam.de until **April 19 (23:59), 2024**. In your email, please:
    - Tell me your name, semester, and major
    - Name your **top-3 paper choices** from the syllabus for presenting
    - Explain why you want to take this course
    - List some of your related experience in deep learning/natural language processing/implementing NLP models

Grading

- Questions about readings: 20%
  - Questions are graded on a 3-point scale (0: no question submitted, 1: superficial question, 2: insightful question).
- Presentation: 30%
  - 1 assigned paper
- Final paper: 50%
  - 5 pages of main content following the ACL template
  - A technical report of a small independent project / A review paper / topic of your choice in discussion with me
  - Proposal due date: **June 16 (23:59), 2024**
  - Final paper due date: **October 13 (23:59), 2024**

Presentation

- Motivate meaningful questions in a context and think about why this paper is important (You're expected to read more papers if you want to fully understand papers)
- Pay attention to technical details, but present experimental results properly
- Highlight take-aways and think about what can be done in the future
- Rehearsal is important, and I am happy to provide comments and give feedback



[Yang, Jingfeng, et al (2023)]()

Topics

Date	Topic	Readings	Related Materials	Presenter	Slides
	Part 1: Introduction				
2024/04/16	Transformer	Attention is All you Need	The Annotated Transformer; The Illustrated Transformer; Huggingface NLP course on transformer	Meng	
2024/04/23	Pretrained Models with Encoder-only Architecture	(1) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding; (2) RoBERTa: A Robustly Optimized BERT Pretraining Approach; (3) ALBERT: A Lite BERT for Self-supervised Learning of Language Representations; (4) SpanBERT: Improving Pre-training by Representing and Predicting Spans; (5) XLNet: Generalized Autoregressive Pretraining for Language Understanding	A Primer in BERTology: What We Know About How BERT Works	Meng	

	Part 2: Model Architecture and Learning		
2024/05/07	Tokenization	(1) Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP; (2) Unpacking Tokenization: Evaluating Text Compression and its Correlation with Model Performance	Neural Machine Translation of Rare Words with Subword Units ; Huggingface NLP course on tokenizers; BPE Explainer
2024/05/14	Self-Supervised Learning	(1) Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks; (2) A Simple Framework for Contrastive Learning of Visual Representations	A Cookbook of Self-Supervised Learning; A Primer on Contrastive Pretraining in Language Processing: Methods, Lessons Learned and Perspectives; Tutorial on SimCLR
2024/05/21	Transfer Learning	(1) CogTaskonomy: Cognitively Inspired Task Taxonomy Is Beneficial to Transfer Learning in NLP; (2) Beto, Bentz, Becas: The Surprising Cross-Lingual Effectiveness of BERT	Tutorial on transfer learning for NLP (NAACL 2019) [code]

	Part 3: Model Analysis and Interpretation		
2024/05/28	Linguistic Knowledge of Pretrained Models	(1) A Structural Probe for Finding Syntax in Word Representations; (2) Probing Pretrained Language Models for Lexical Semantics	Probing Classifiers: Promises, Shortcomings, and Advances; Designing and Interpreting Probes with Control Tasks
2024/06/04	World Knowledge of Pretrained Models	(1) Evaluating Commonsense in Pre-Trained Language Models; (2) Probing Pre-Trained Language Models for Cross-Cultural Differences in Values	

	Part 4: Efficient Pretrained Models		
2024/06/11	Pruning	(1) The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks; (2) Are Sixteen Heads Really Better than One?	Compressing Large-Scale Transformer-Based Models A Case Study on BERT; Pytorch pruning tutorial; Diving Into Model Pruning in Deep Learning
2024/06/18	Quantization	(1) Understanding and Overcoming the Challenges of Efficient Transformer Quantization; (2) I-BERT: Integer-only BERT Quantization	Pytorch quantization recipe; A Tale of Model Quantization in TF Lite
2024/06/25	Knowledge Distillation	(1) TinyBERT Distilling BERT for Natural Language Understanding; (2) MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers	Distilling the Knowledge in a Neural Network; Pytorch knowledge distillation tutorial; Distilling Knowledge in Neural Networks With Weights & Biases

Discussion

- What are you most excited about pretrained models and want to learn from this class?

Transformer

- Recap
  - MT: RNN+attention -> Transformer:
  - Huggingface transformer library **Hugging Face**
- Learning tips
  - Build transformers on your own ([The annotated transformer by Alexander Rush]())
  - Multi 30K dataset, etc.

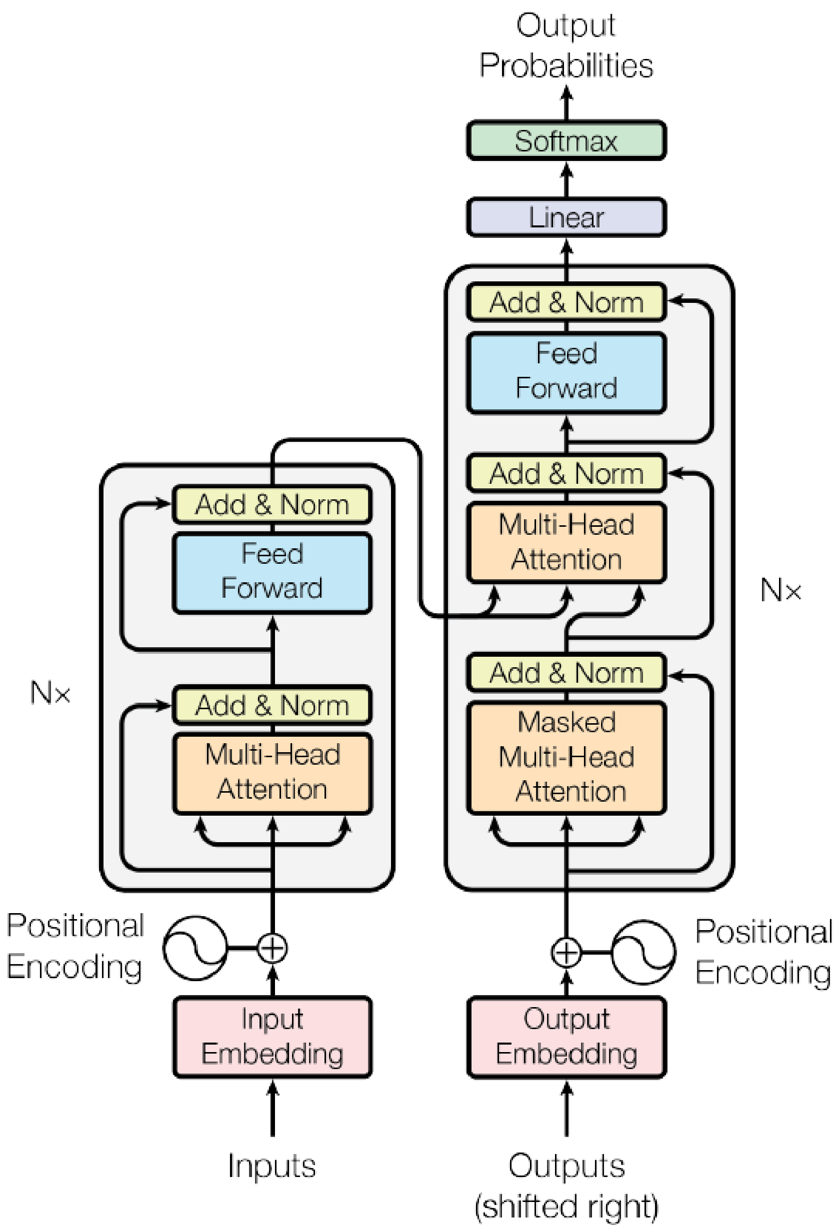


Figure 1: The Transformer - model architecture.

[Vaswani et al (2017)]()

Transformer

- Encoder-decoder architecture;
- Token Embedding & Positional Encoding;
- Residual Connection & Layer Normalization;
- Multi-head Attention;
- Position-wise Feed-forward Networks;

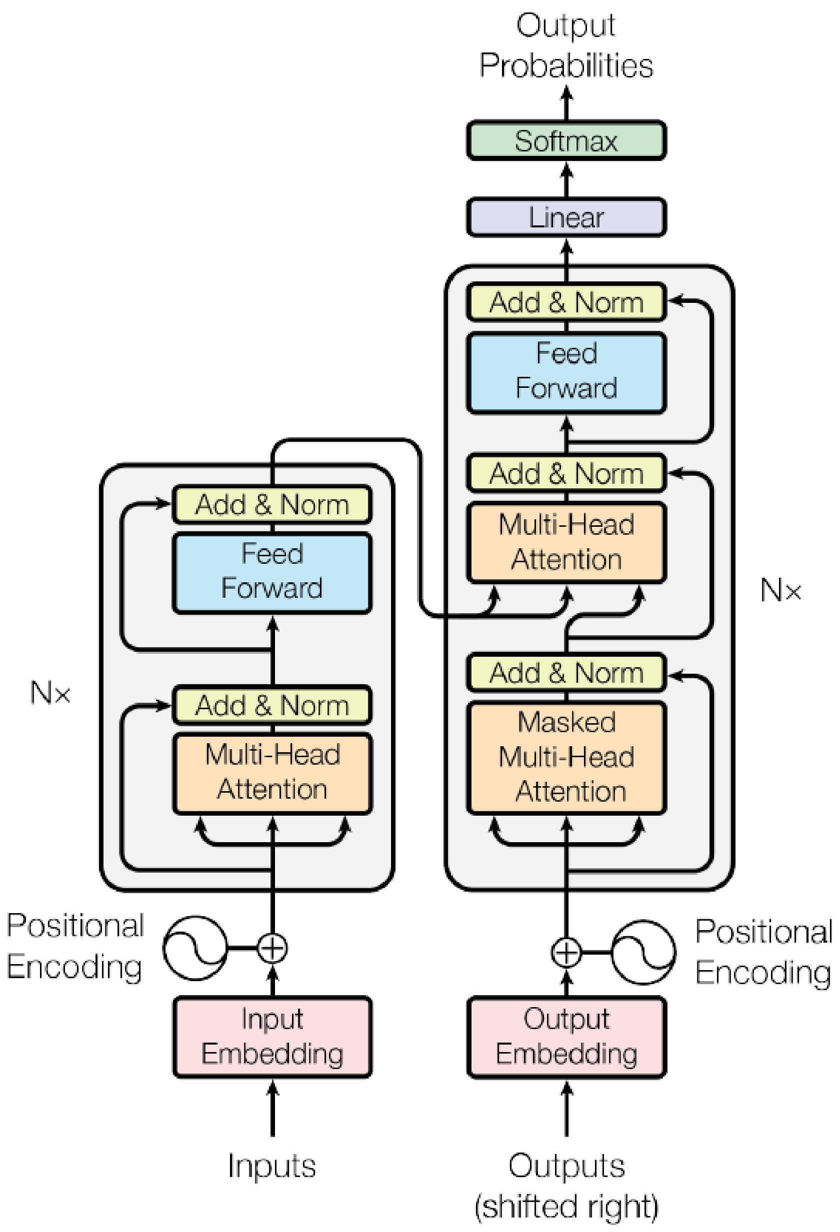


Figure 1: The Transformer - model architecture.

Vaswani et al (2017)

Token Embedding

- Token embeddings are multiplied by a scaling factor `sqrt(d_model)`, where `d_model` is the hidden dimension size.

```
class Embeddings(nn.Module):
    def __init__(self, d_model, vocab):
        super(Embeddings, self).__init__()
        self.lut = nn.Embedding(vocab, d_model)
        self.d_model = d_model

    def forward(self, x):
        return self.lut(x) * math.sqrt(self.d_model)
```

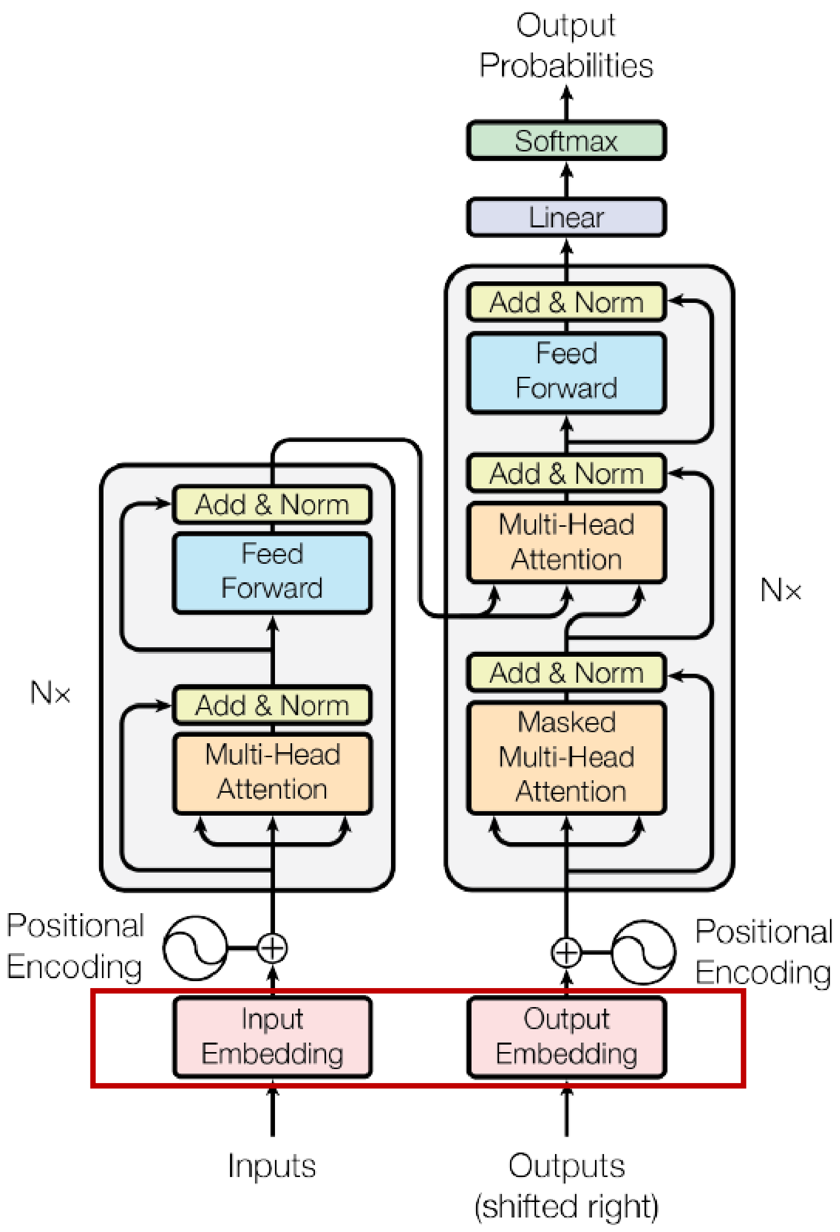


Figure 1: The Transformer - model architecture.

[Vaswani et al (2017)]

Positional Encoding

- Original transformer paper use fixed static encoding; BERT use learnable embedding.
- The positional encodings have the same dimension d_model as the embeddings, so that the two can be summed.

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

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

- where *pos* is the position and *i* is the dimension.

```
# Compute the positional encodings once in log space.
pe = torch.zeros(max_len, d_model)
position = torch.arange(0, max_len).unsqueeze(1)
div_term = torch.exp(
    torch.arange(0, d_model, 2) * -(math.log(10000.0) / d_model)
)
pe[:, 0::2] = torch.sin(position * div_term)
pe[:, 1::2] = torch.cos(position * div_term)
pe = pe.unsqueeze(0)
```

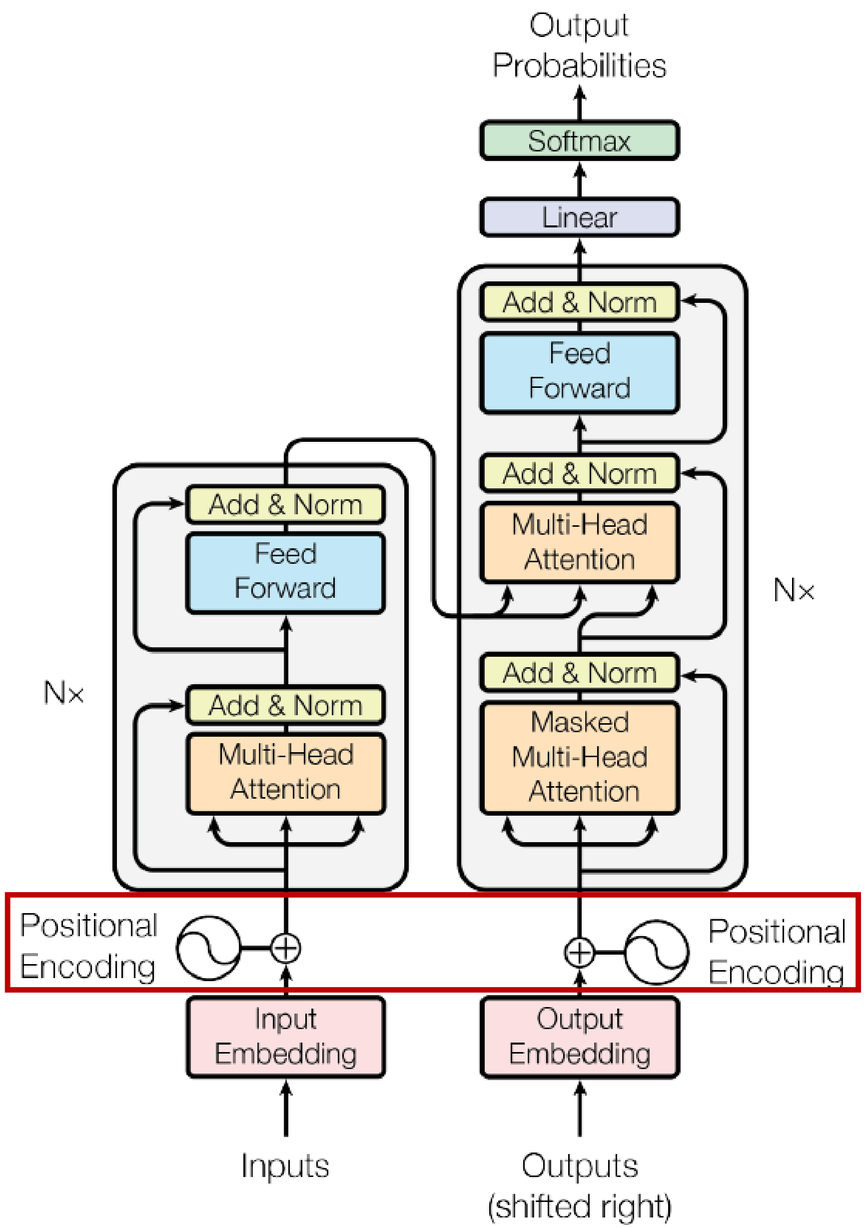


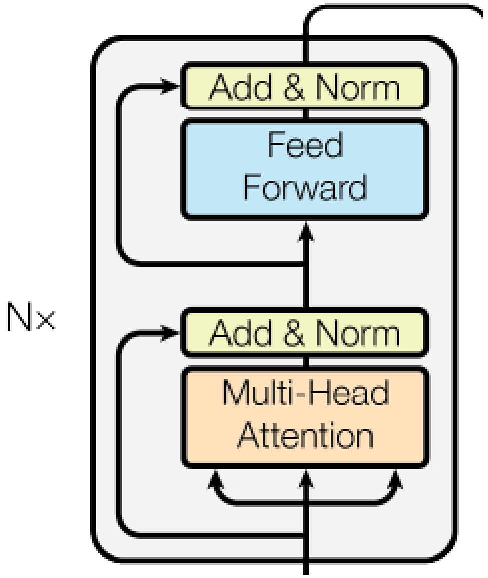
Figure 1: The Transformer - model architecture.

Vaswani et al (2017)

Encoding Layer

- Sublayer 1: Multi-head attention -> Residual connections and layer normalization;
- Sublayer 2: Position-wise feed-forward networks -> Residual connections and layer normalization;

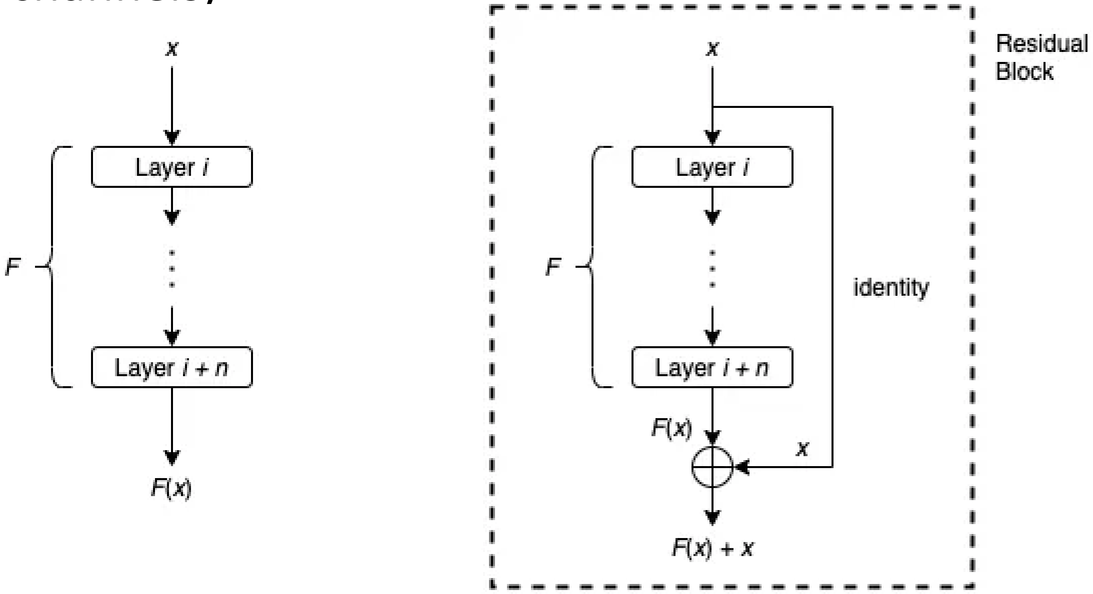
```python
class EncoderLayer(nn.Module):
    "Encoder is made up of self-attn and feed forward (defined below)"

    def __init__(self, size, self_attn, feed_forward, dropout):
        super(EncoderLayer, self).__init__()
        self.self_attn = self_attn
        self.feed_forward = feed_forward
        self.sublayer = clones(SublayerConnection(size, dropout), 2)
        self.size = size

    def forward(self, x, mask):
        "Follow Figure 1 (left) for connections."
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, mask))
        return self.sublayer[1](x, self.feed_forward)
```



Vaswani et al (2017)

Residual connections

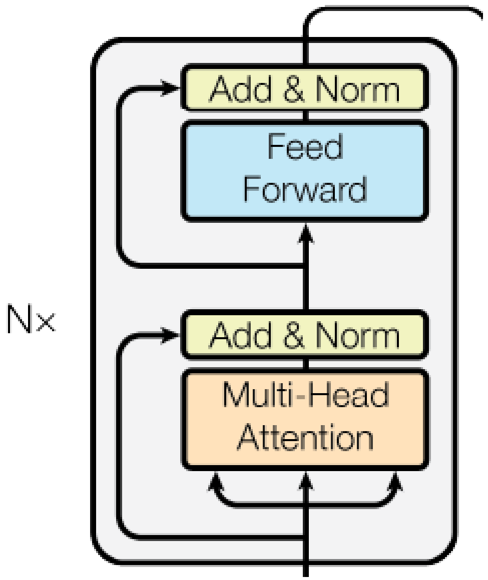
- a simple yet very effective technique to make training deep neural networks easier;
- ResNet 1000 layers
- A very deep network may act like a combination of shallower networks (with more channels)



Traditional Feedforward without Residual Connection

With Residual Connection

[Source]

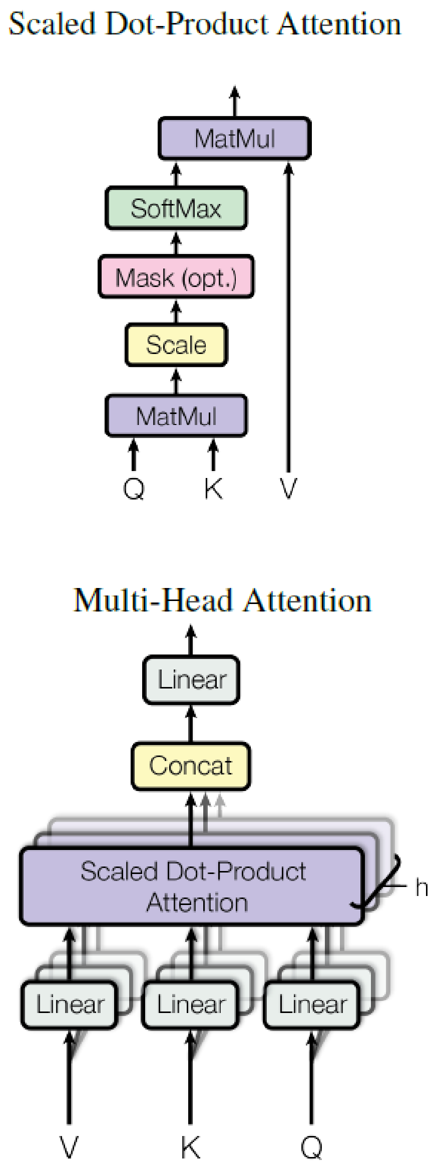


[Vaswani et al (2017)]

Attention

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

- How to understand this formula? Do not scared by Q, K, V matrix computations …
- What is the inner product of vectors, how is it calculated, and most importantly, what is its geometric interpretation?
- If we multiply a matrix **X** by its own transpose, what does it mean?



[Vaswani et al (2017)](https://arxiv.org/abs/1706.03762)

Attention

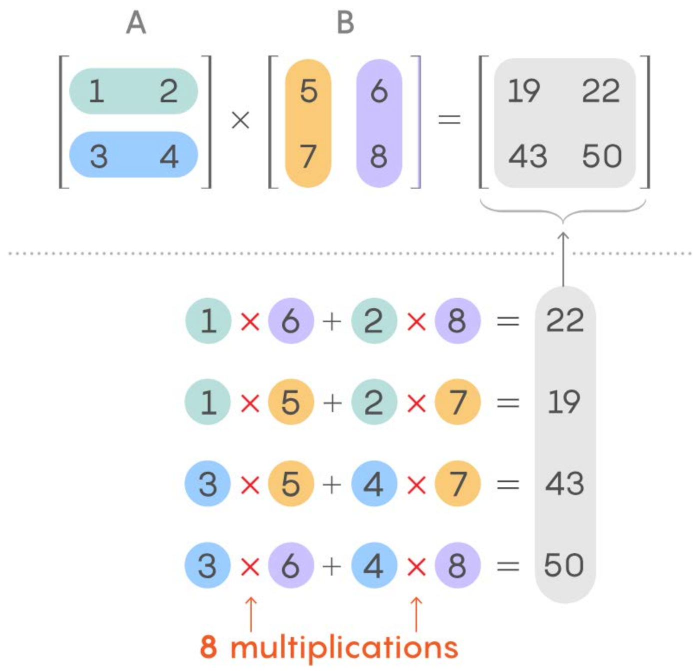
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

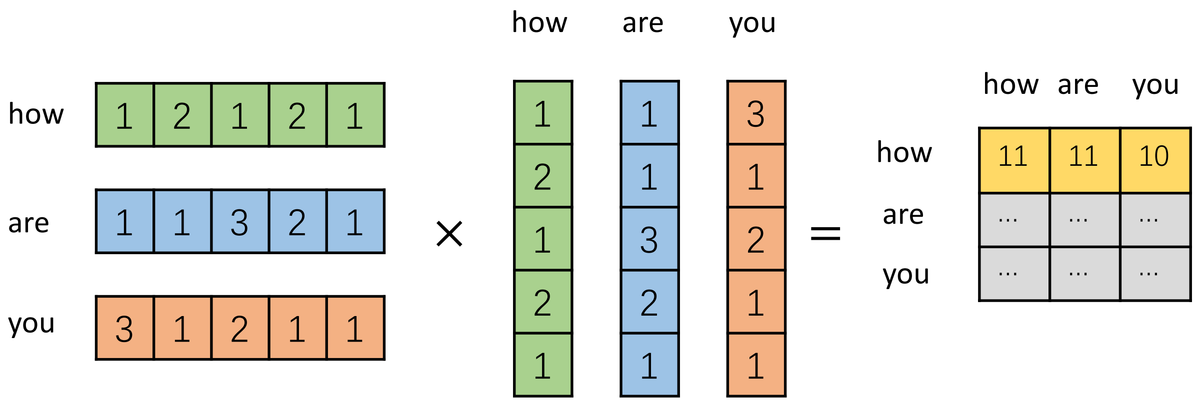
- $\boldsymbol{softmax(XX^T)X}$
  - What does $\boldsymbol{XX^T}$ stand?

Attention

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

- $\boldsymbol{softmax(XX^T)X}$
  - What does $\boldsymbol{XX^T}$ stand?
  - The geometric meaning of the inner product of vectors? The angle between two vectors and the projection of one vector on another vector.
  - The larger the value of the projection, the higher the correlation between the two vectors. (when paying attention to word *how*, more attention also should be given to word *are*)
  - XX^T is a square matrix that stores the result of the inner product operation of each vector with itself and other vectors.

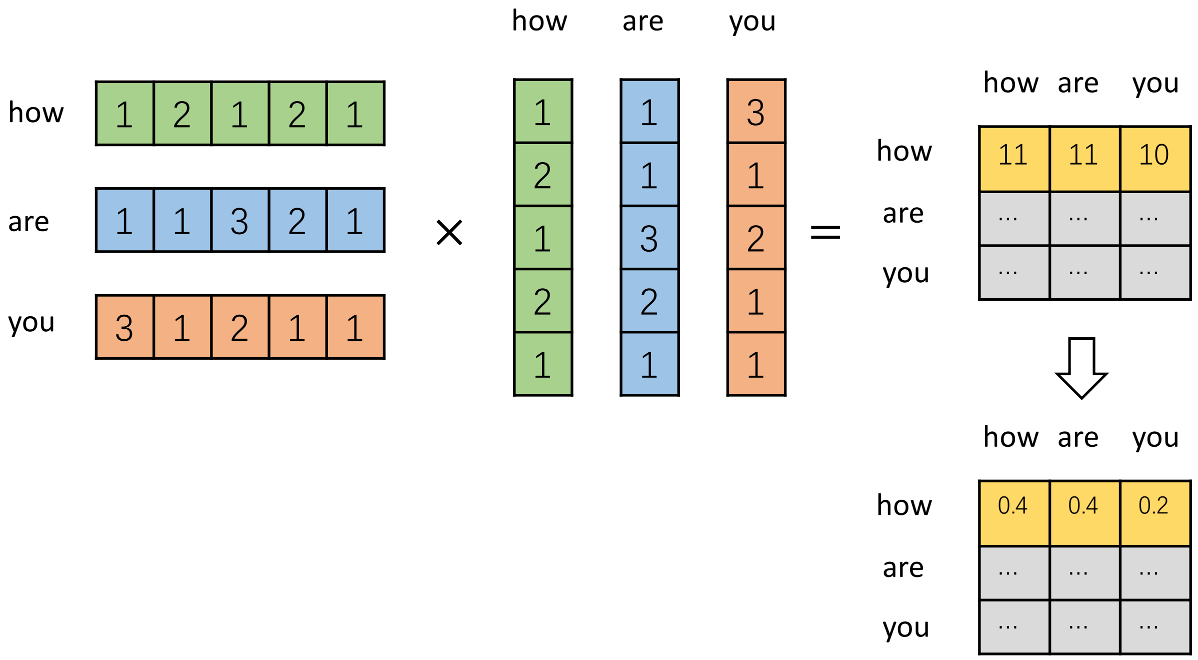




Attention

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

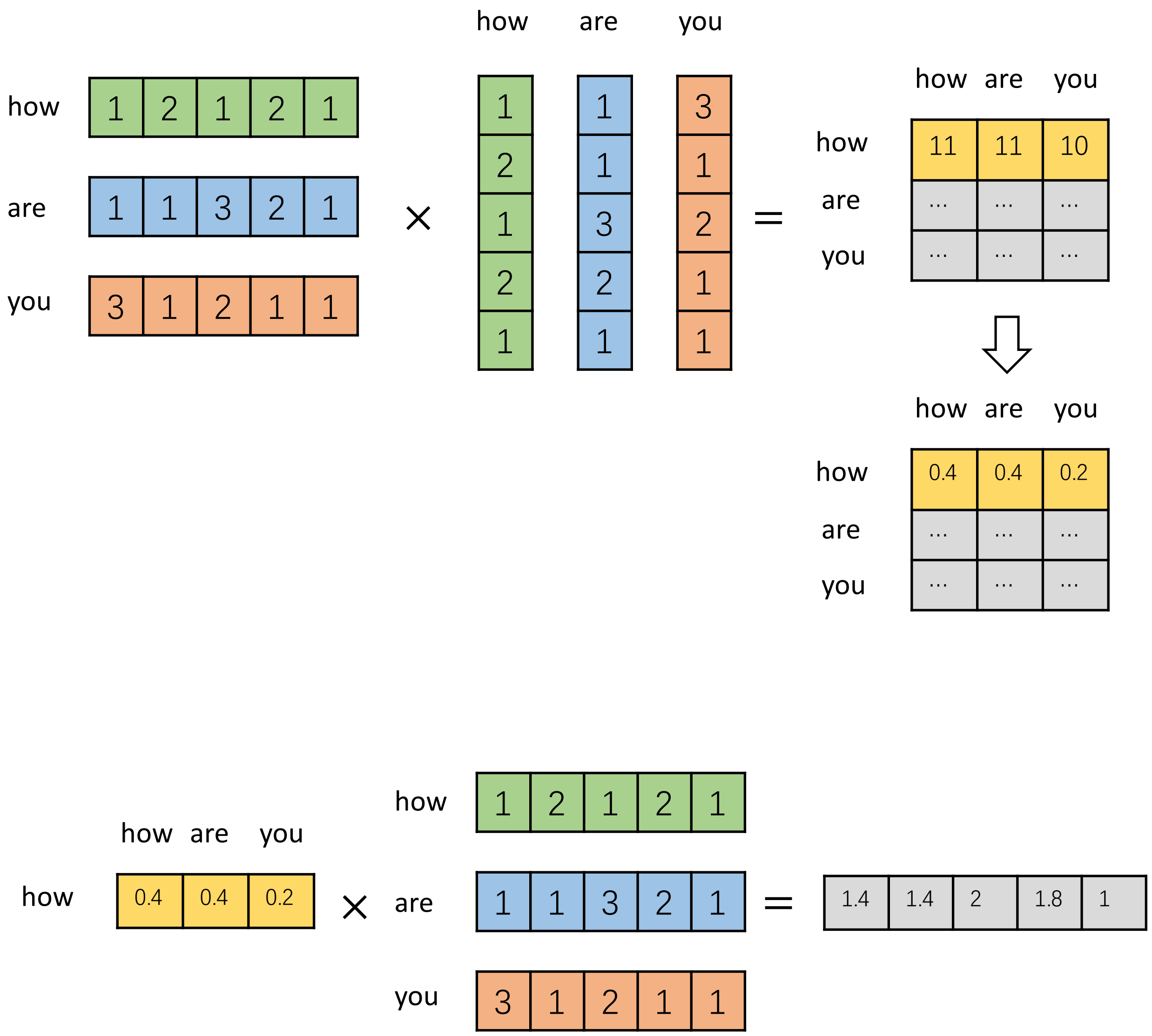
- $\boldsymbol{softmax(XX^T)X}$
  - What about adding **softmax** function?
  - Normalization (the sum of these numbers is 1)
  - what is the core of the Attention mechanism?
  - Weighted sum. Weights are the numbers after normalization. When we focus on the word “how", we should allocate 0.4 of our attention to it itself, leaving 0.4 to focus on “are" and 0.2 to “you".



Attention

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

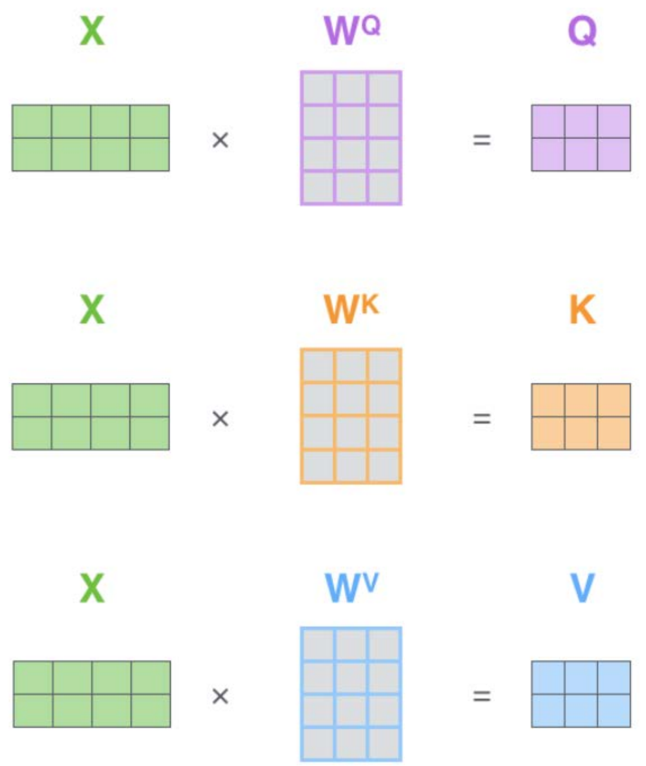
- $\boldsymbol{softmax(XX^T)X}$
  - What is the meaning of the last $\boldsymbol{X}$?
  - The new vector is the weighted sum of the “how" word vectors through the attention mechanism



Attention

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

- What are Q, K, V exactly?
- Q, K, V are derived from the product of X and matrix W, which are essentially linear transformation of X.
- Why not just use X but linearly transform it?
- Trainable W matrices could improve the fitting capacity of the model.



[The illustrated transformer]

Attention

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

- $\sqrt{d_k}$
- Dimensions of Q, K -> the variance of $softmax(QK^T)$ and stable training

```
def attention(query, key, value, mask=None, dropout=None):
    "Compute 'Scaled Dot Product Attention'"
    d_k = query.size(-1)
    scores = torch.matmul(query, key.transpose(-2, -1)) / math.sqrt(d_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)
    p_attn = scores.softmax(dim=-1)
    if dropout is not None:
        p_attn = dropout(p_attn)
    return torch.matmul(p_attn, value), p_attn
```

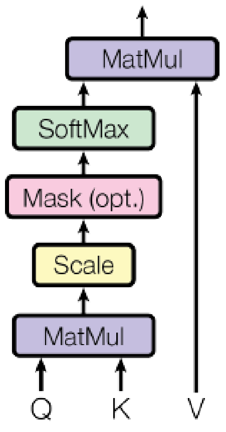
Attention

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

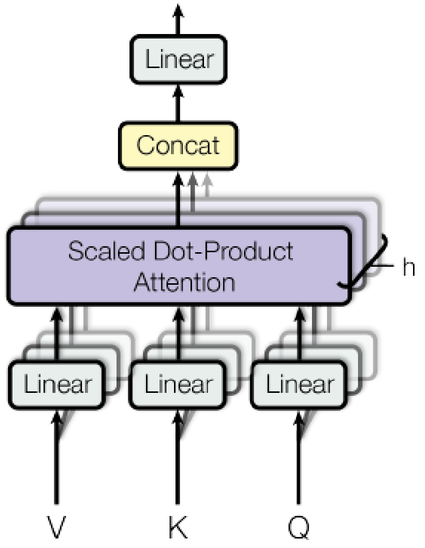
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_\text{model}}$.

Scaled Dot-Product Attention



Multi-Head Attention

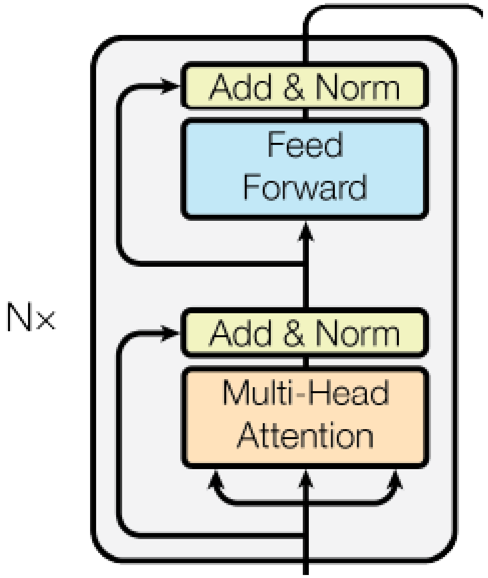


[Vaswani et al (2017)]

Position-wise Feed-forward Networks

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

```python
class PositionwiseFeedForward(nn.Module):
    "Implements FFN equation."

    def __init__(self, d_model, d_ff, dropout=0.1):
        super(PositionwiseFeedForward, self).__init__()
        self.w_1 = nn.Linear(d_model, d_ff)
        self.w_2 = nn.Linear(d_ff, d_model)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        return self.w_2(self.dropout(self.w_1(x).relu()))
```

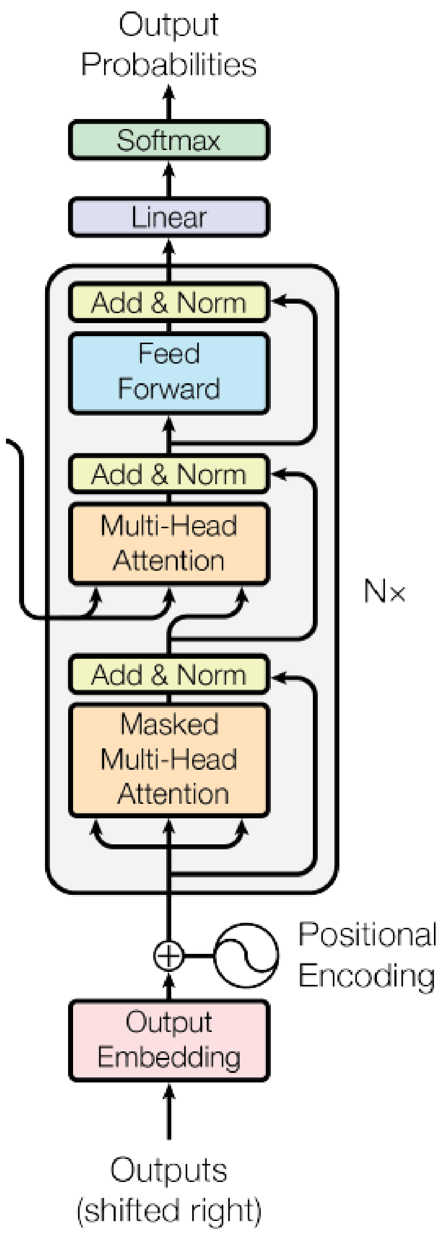


[Vaswani et al (2017)]

Decoding Layer

- masked multi-head attention layer: the decoder representation so far as the query, key and value (target mask to prevent peaking/cheating);
- multi-head attention layer: the decoder representation as the query and the encoder representation as the key and value;

```python
class DecoderLayer(nn.Module):
    "Decoder is made of self-attn, src-attn, and feed forward (defined below)"

    def __init__(self, size, self_attn, src_attn, feed_forward, dropout):
        super(DecoderLayer, self).__init__()
        self.size = size
        self.self_attn = self_attn
        self.src_attn = src_attn
        self.feed_forward = feed_forward
        self.sublayer = clones(SublayerConnection(size, dropout), 3)

    def forward(self, x, memory, src_mask, tgt_mask):
        "Follow Figure 1 (right) for connections."
        m = memory
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, tgt_mask))
        x = self.sublayer[1](x, lambda x: self.src_attn(x, m, m, src_mask))
        return self.sublayer[2](x, self.feed_forward)
```



[Vaswani et al (2017)](#)

Not covered

- Source / target masking
- Optimization / regularization:
  - Layer normalization;
  - Label smoothing
- Training settings
- MT: decoding methods (greedy, beam search, etc.)

Bonus

- Attention in transformers, visually explained