Deep Learning 1

2025-2026 - Pascal Mettes

Lecture 6

Attention-based Deep Learning

Previous lecture

Lecture	Title
1	Intro and history of deep learning
3	Deep learning optimization I
5	Convolutional deep learning
7	Graph deep learning
9	Multi-modal deep learning
11	What doesn't work in deep learning
13	Q&A

Lecture	Title
2	AutoDiff
4	Deep learning optimization II
6	Attention-based deep learning
8	From supervised to unsupervised deep learning
10	Generative deep learning
12	Non-Euclidean deep learning
14	Deep learning for videos

This lecture

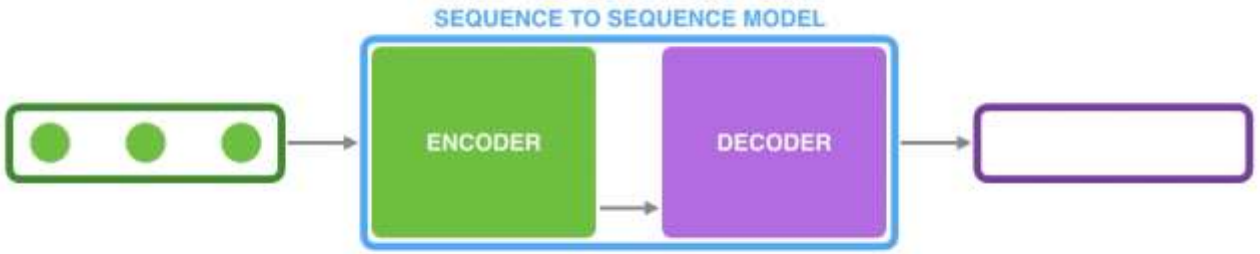
Sequential modelling

Attention and self-attention

Transformers

Language and vision transformers

Sequence2sequence models

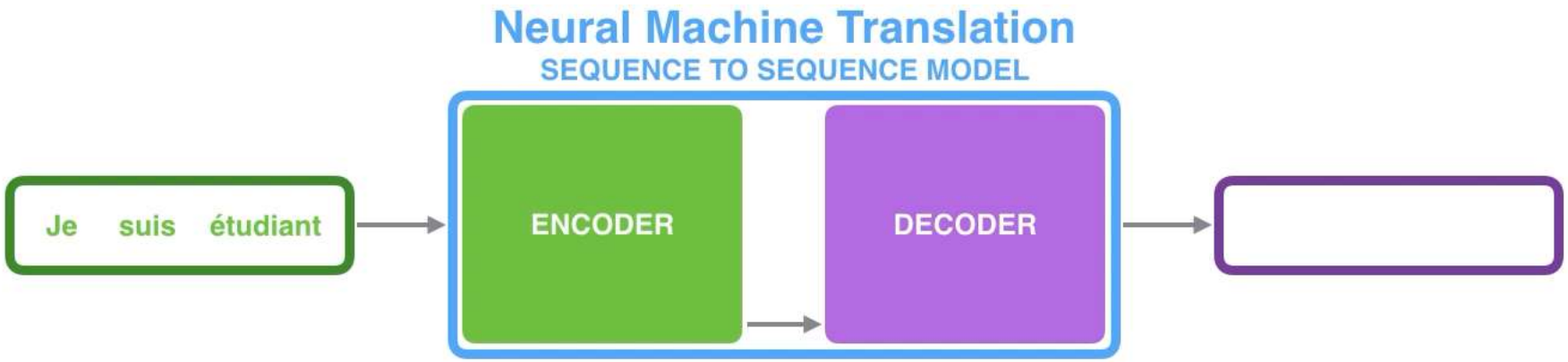


Encoder-decoder models with 2 parts:

1. **Encoder:** takes a variable-length sequence of elements as the input and transforms it into a context representation with a fixed-size.
2. **Decoder:** maps the encoded state of a fixed size to a variable-length sequence of elements.

Was a backbone for many problems, especially in language

Sequence2sequence example



Encoder: reads sentence in one language and converts to context vector.

Decoder: outputs a translation from context vector.

Sequence2sequence example: encoder

An encoder encodes the input sentence, a sequence of vectors $x = (x_1, x_2, \dots x_{Tx})$, into a context vector $\mathbf{c}$.

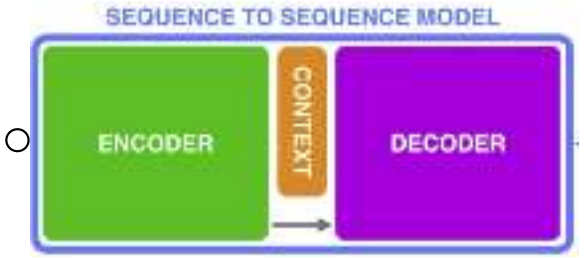
A common approach is to use an RNN/LSTM, such that:

$$h_t = f(x_t, h_{t-1}),$$
$$c = q(\{h_1, \dots h_{Tx}\}).$$

$\mathbf{h_t} \in \mathbb{R}$ is a hidden state at time-step t,

c is the context vector generated from the sequence o states,

$\mathbf{f}$ and $\mathbf{q}$ are nonlinear functions.



$h_t = f(x_t, h_{t-1})$ $c = q(\{h_1, \dots h_{Tx}\})$

Sequence2sequence example: decoder

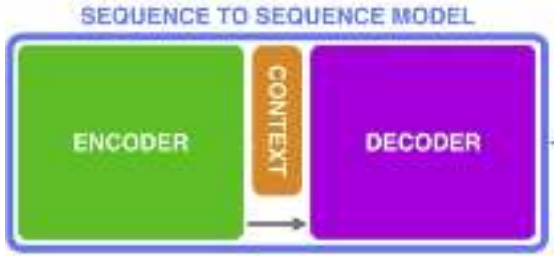
The decoder is trained to predict the next word y_t, given the context vector c and all the previously predicted words $\{y_1, \ldots, y_{t-1}\}$.

It defines a probability over the translation y by decomposing the joint probability into the conditionals:

$$p(y) = \prod_{t=1}^{T} p(y_t \mid \{y_1, \ldots, y_{t-1}\}, \mathbf{c}), \text{ where } y = (y_1, \ldots, y_{Ty}).$$

With an RNN decoder, each conditional probabilty is modeled as:

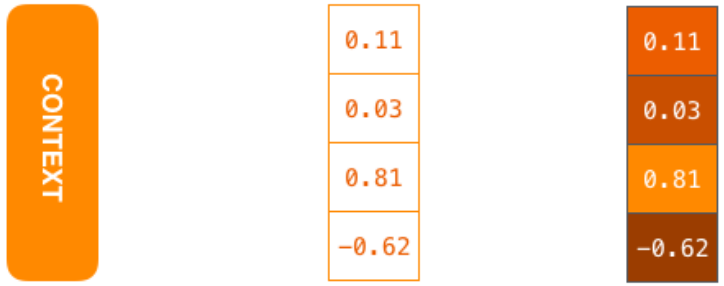
$$p(y_t \mid \{y_1, \ldots, y_{t-1}\}, \mathbf{c}) = g(y_{t-1}, s_t, \mathbf{c}), \text{ where}$$



g is a nonlinear (multi-layered) function $h_t = f(x_t, h_{t-1})$ $c = q(\{h_1, \ldots h_{Tx}\})$

Limitation of seq2seq models

The model needs to compress all necessary information of a source sequence into the fixed-length context vector c, i.e., there is a bottleneck.



The context is a vector of floats, and its size is the number of hidden units in the encoder RNN.

When dealing with long sequences, it is challenging for the model to compress all information into a fixed-

A simple solution: attention

Attention strives to overcome the bottleneck issue of the encoder-decoder.

Allows the model to focus on all relevant inputs to decode the output.

Neural Machine Translation example: for every new word to generate, the model searches for a set of positions in the input where the most relevant information is concentrated.

A huge breakthrough in language processing, and later in all other domains.

Attention

Stop encoding the whole input in a fixed-length vector.

Instead encode the input into a sequence of vectors.

Then focus on a subset of these vectors adaptively while decoding the output.

I.e., simply stop squashing all information!

Attention formally

Now each conditional probability of the decoder is defined as:

$$p(y_t|\{y_1, \ldots, y_{t-1}\}, x) = g(y_{t-1}, s_i, c_i), \text{ where}$$

s_i is a hidden state for time i, computed by:

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

The probability is conditioned on a distinct context vector c_i for each target word y_i (unlike the conventional encoder-decoder).

The context vector c_i depends on a sequence of annotations $(h_1, \ldots, h_{t-1})$ to which an encoder maps the input sentence.

Each annotation h_i contains information about the whole

Attention formally

The context vector c_i is computed as a *weighted sum* of these annotations h_j:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j ,$$

where the weight α_{ij} of each annotation h_j is computed by the probability:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

where $e_{ij} = a(s_{i-1}, h_j)$ is an alignment model,

It scores how well the inputs around position j and the output at position i match.

The alignment model a is parametrized as a feedforward

Attention formally

The weight α_{ij} reflects the importance of the annotation h_j, with respect to the previous hidden state s_{i-1} when deciding the next state s_i and generating y_i.

This implements a mechanism of attention in the decoder.

The decoder decides which parts of the source sentence to pay attention to.

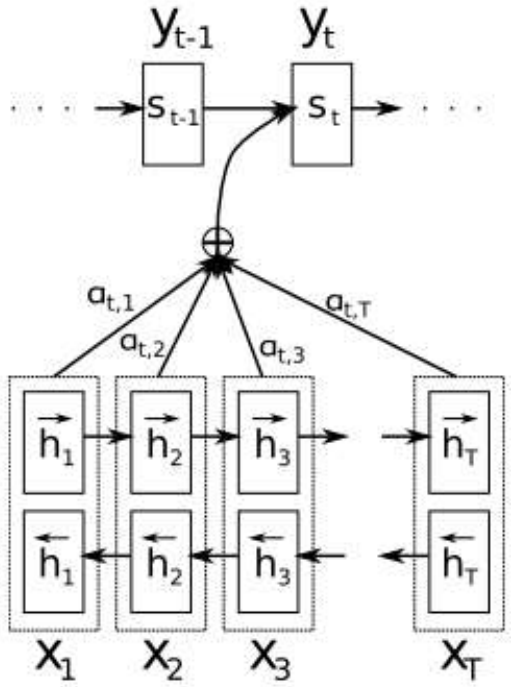
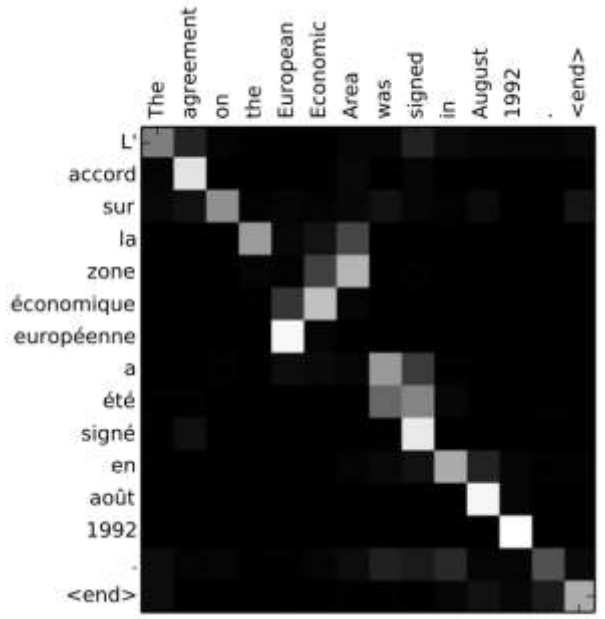


Fig. 1 Graphical illustration of attention model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$[1]

Visualizing attention

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/



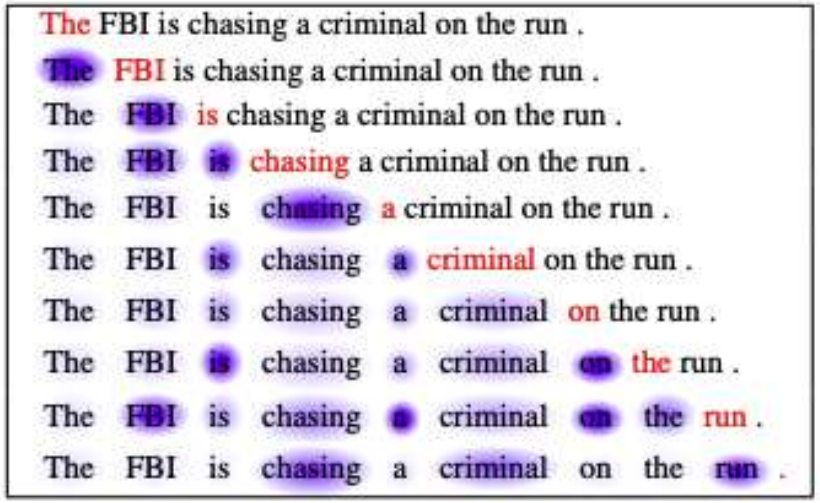
The x-axis and y-axis correspond to the words in the source sentence (English) and the generated translation (French). Each pixel shows the weight α_{ij} of the annotation of the j-th source word for the i-th target word.

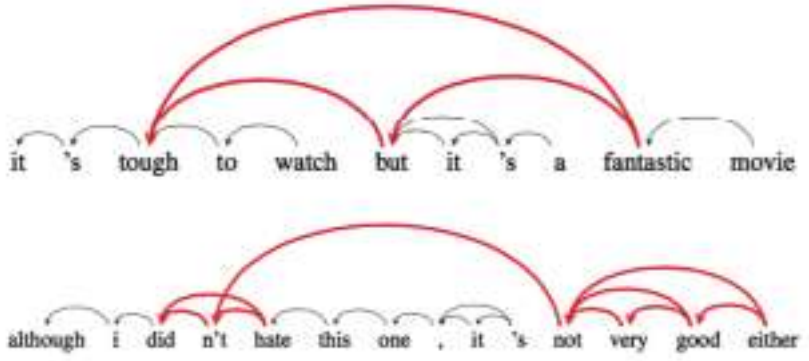
Self-attention

Self-attention (or intra-attention): relates parts of sequence with each other.

Results is a representation of the whole sequence.

In general terms, it can be seen as an operation on sets of elements.





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

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

The transformer

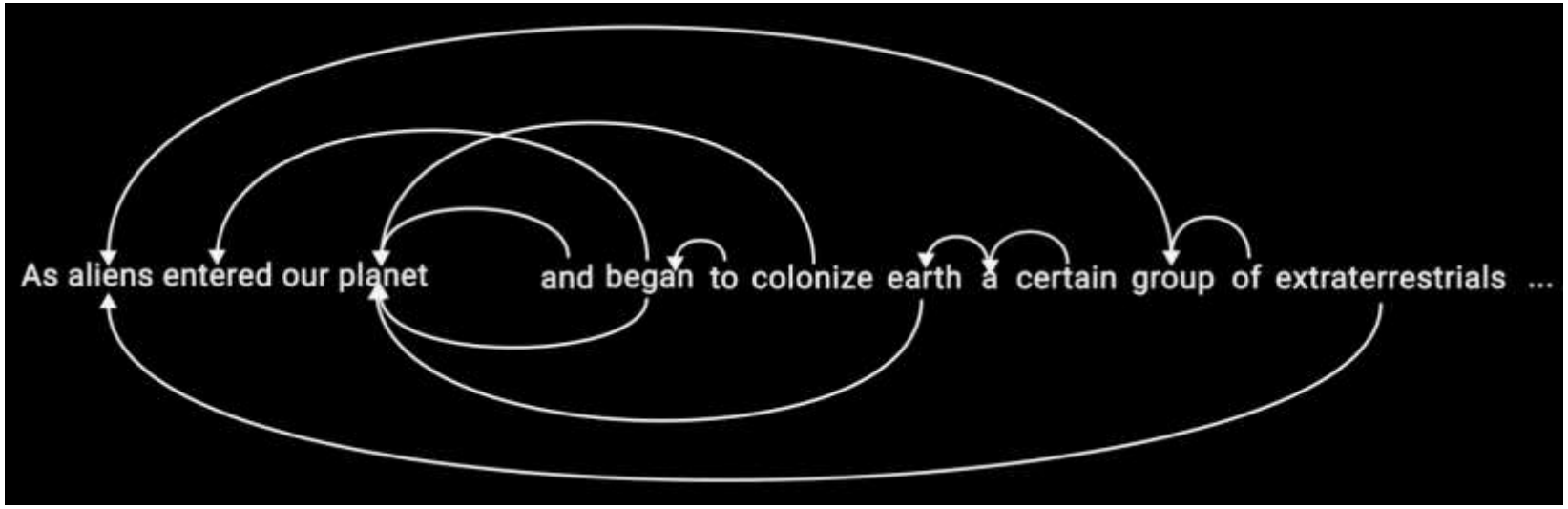
An encoder-decoder model based on (self-)attention mechanisms:

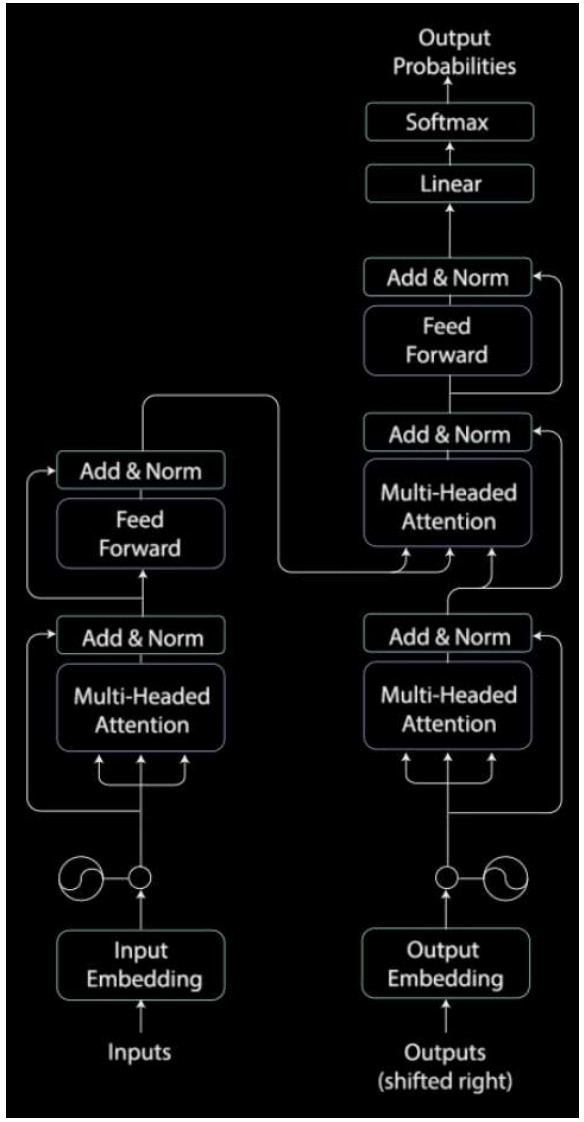
- Without any recurrence and convolutions.
- Referred to as NLP's ImageNet moment.
- It completely changed the landscape of deep learning models.
- Dominates modern deep learning.
- Key behind LLMs.

Important concepts:

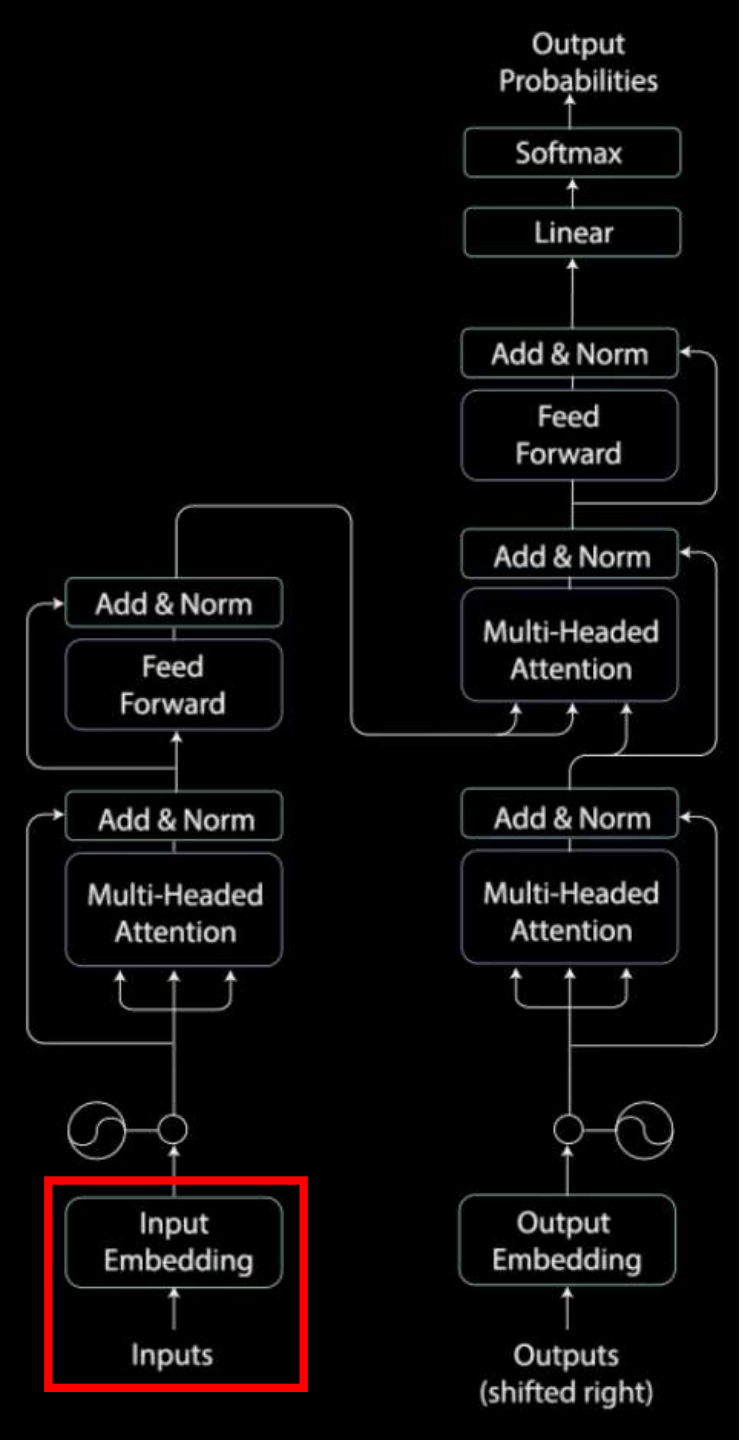
- Queries, keys, values.
- Scaled dot-product attention.
- Multi-head (self-)attention.

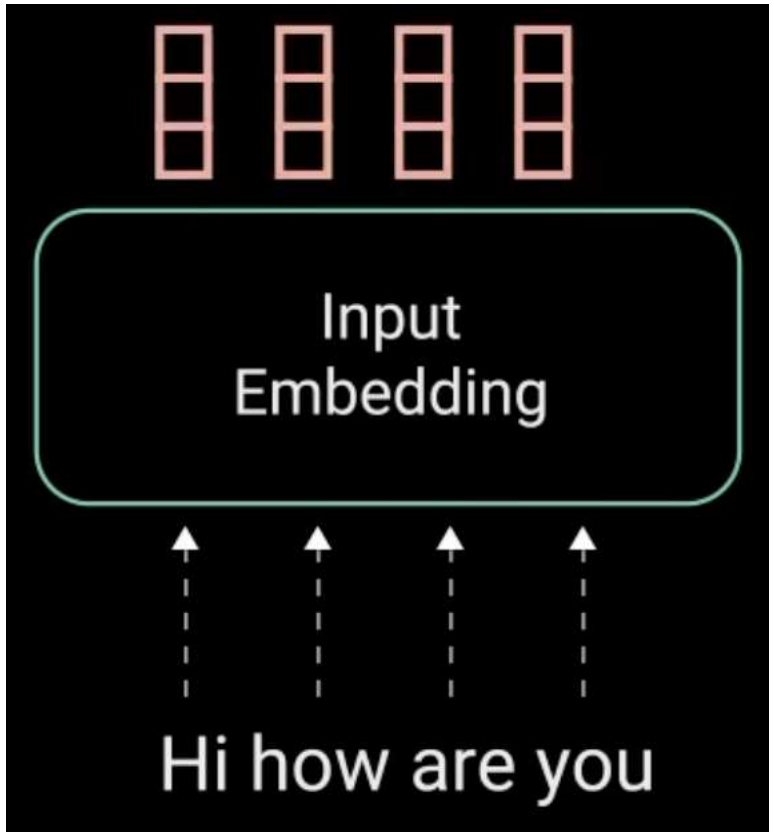
Understanding transformers: let's do a forward pass

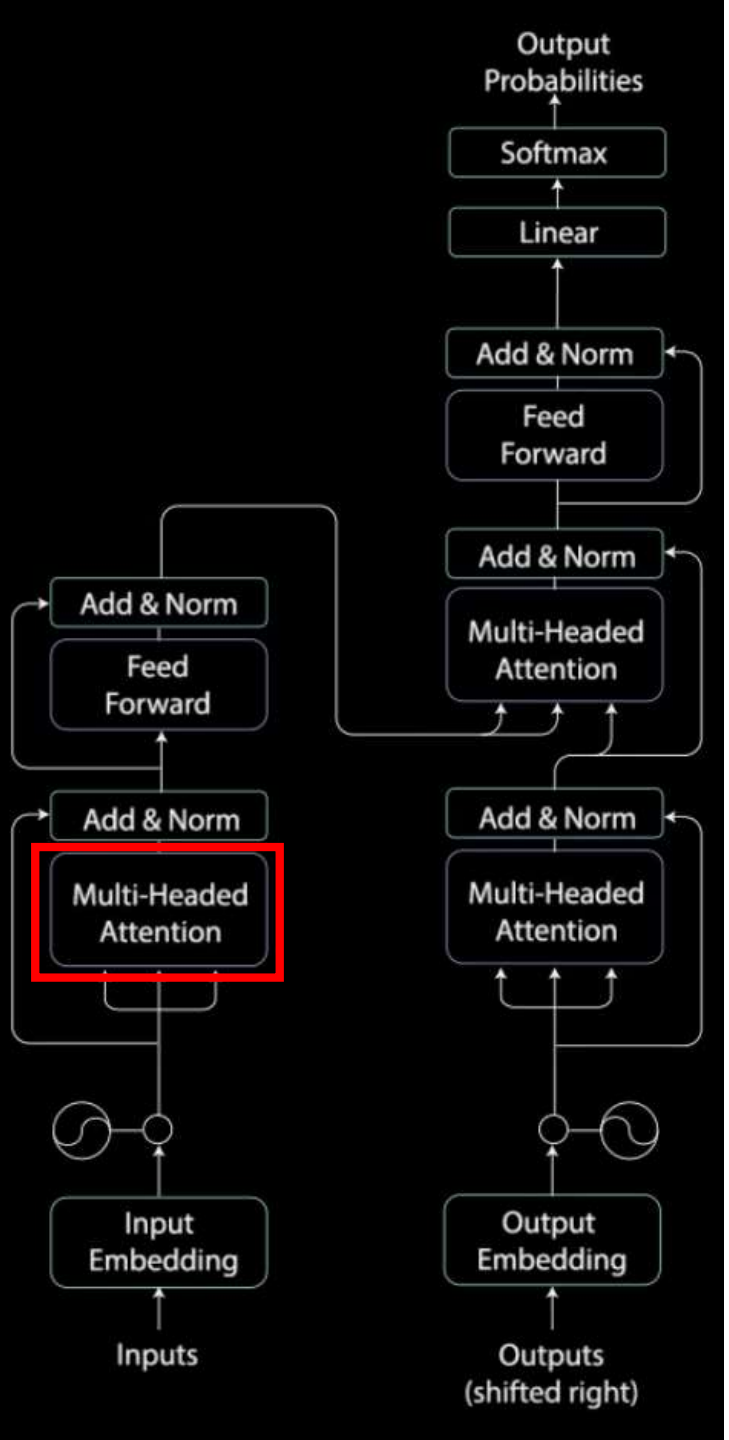


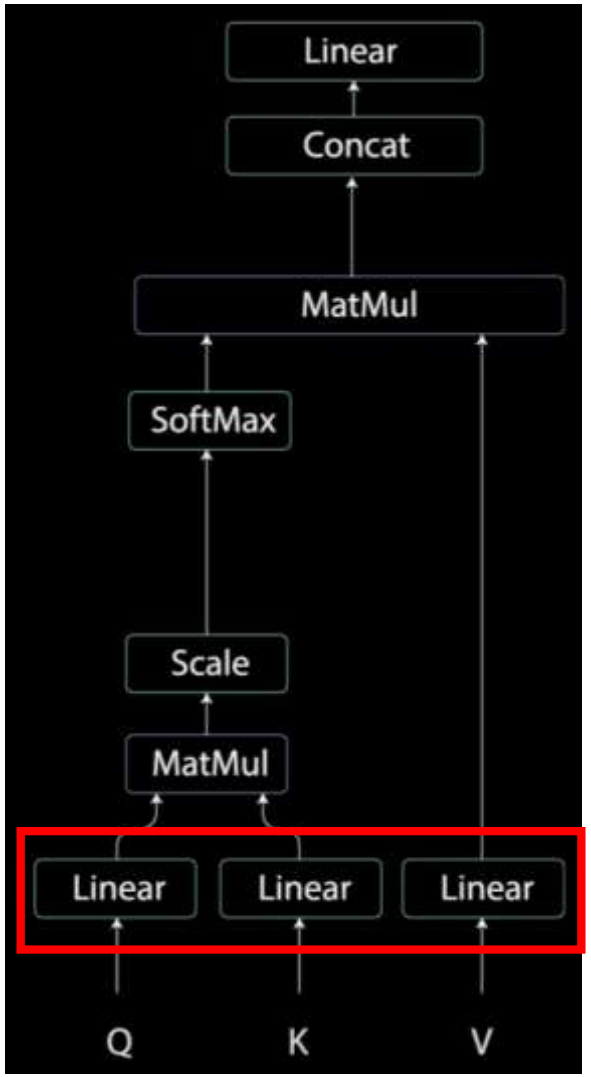


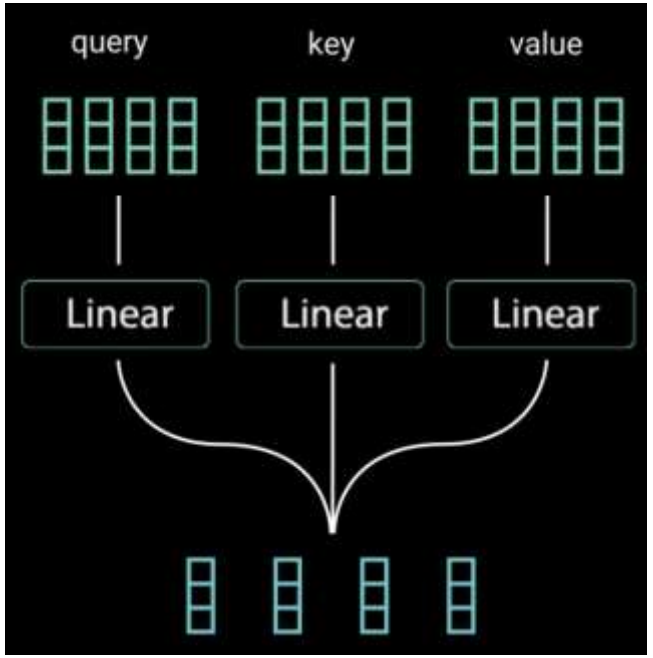
Thank you to Michael Phi for the

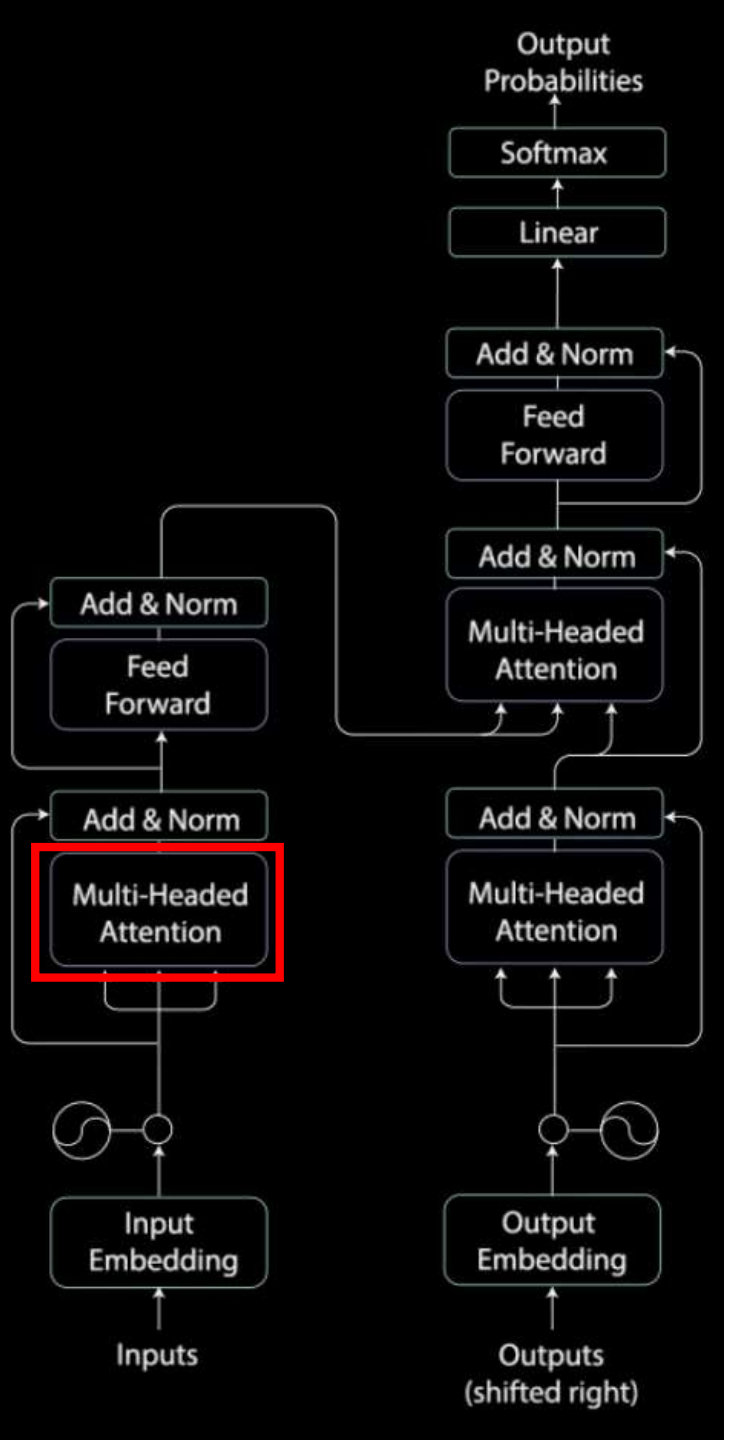

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Headed Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Headed Attention
Add & Norm
Multi-Headed Attention
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

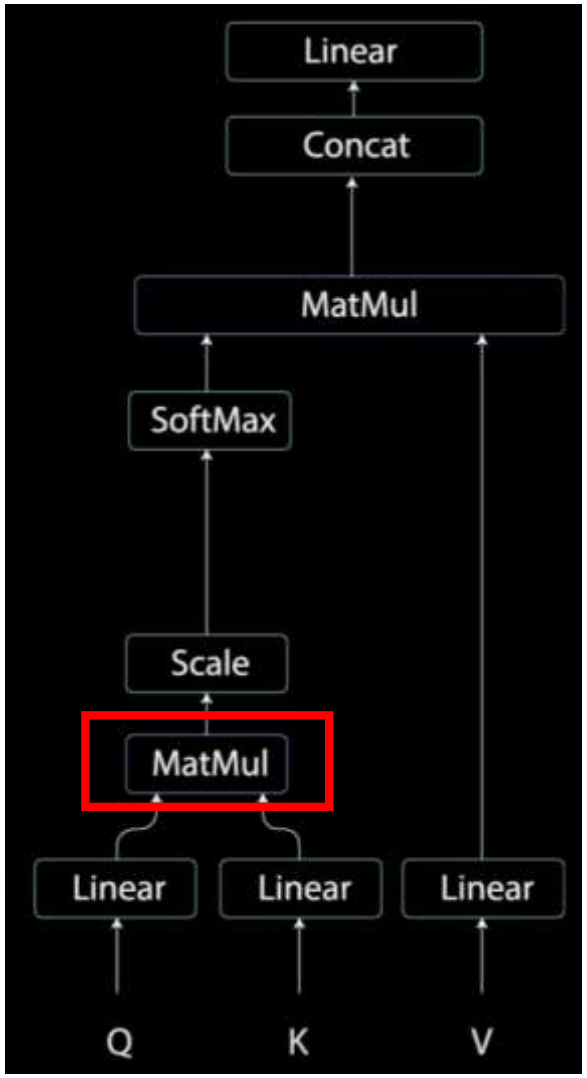

Input Embedding
Hi how are you

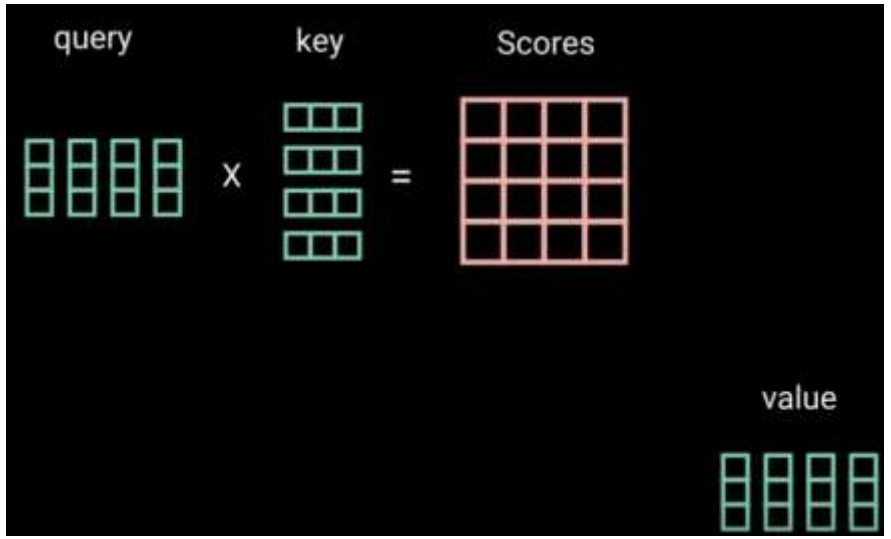

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Headed Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Headed Attention
Add & Norm
Multi-Headed Attention
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)


Linear
Concat
MatMul
SoftMax
Scale
MatMul
Linear
Linear
Linear
Q
K
V

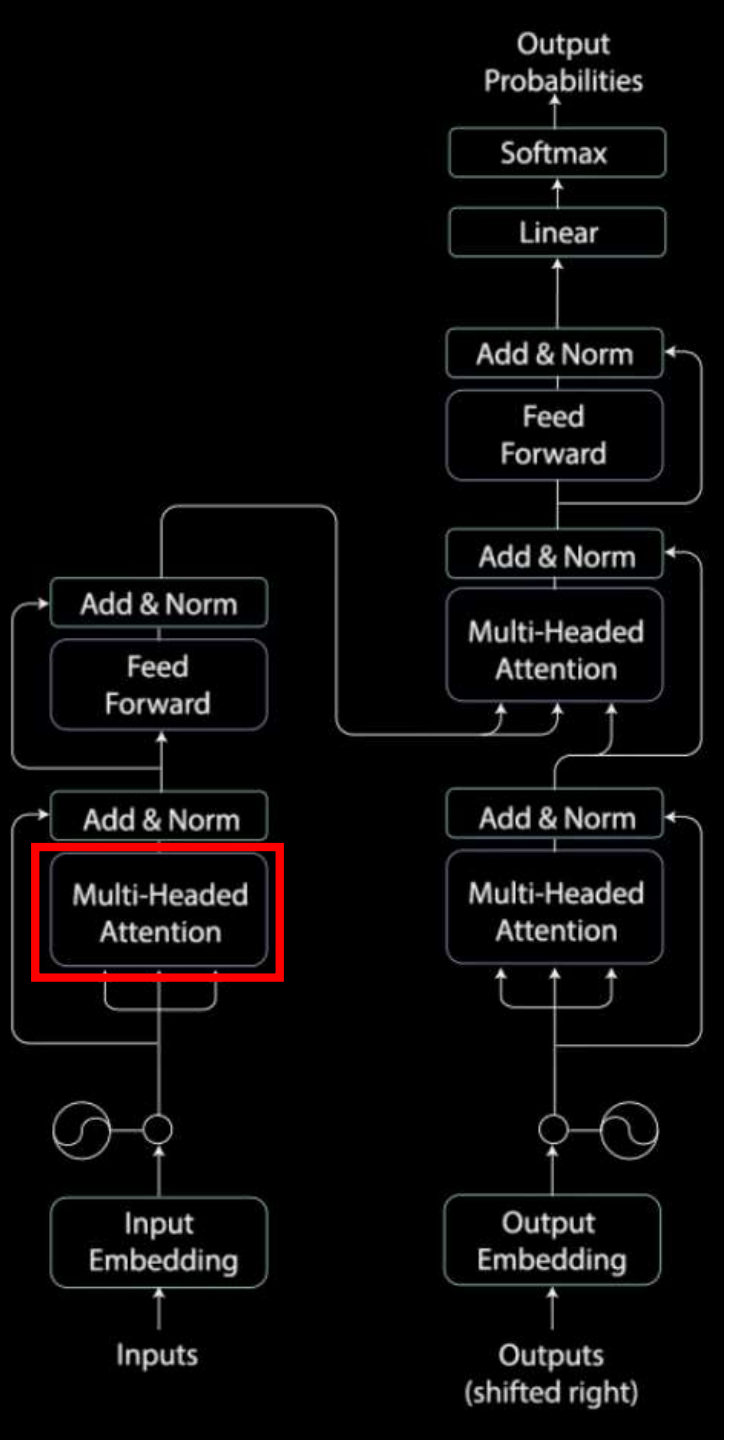

query
key
value
Linear
Linear
Linear

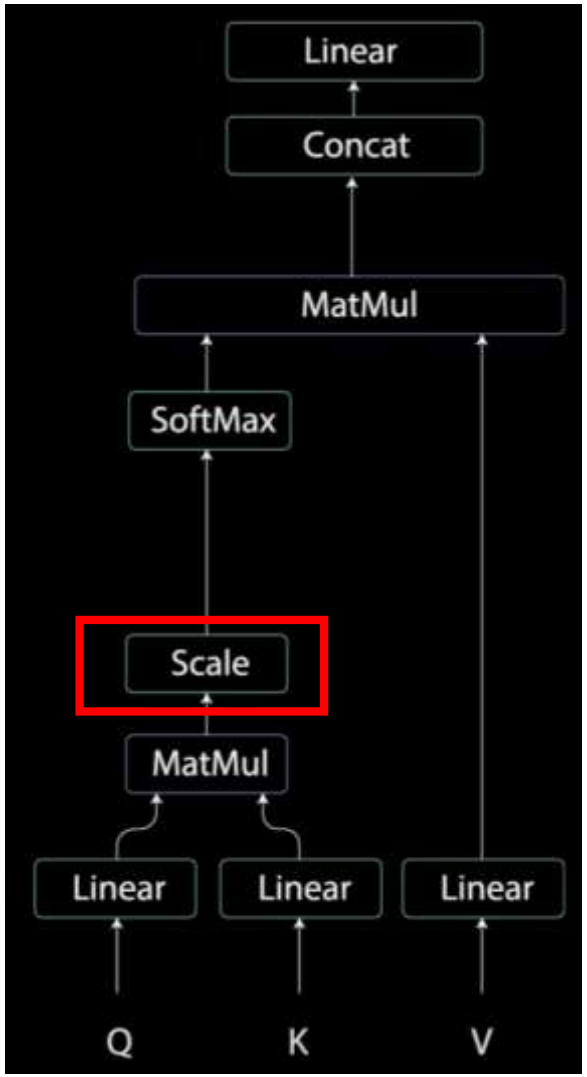


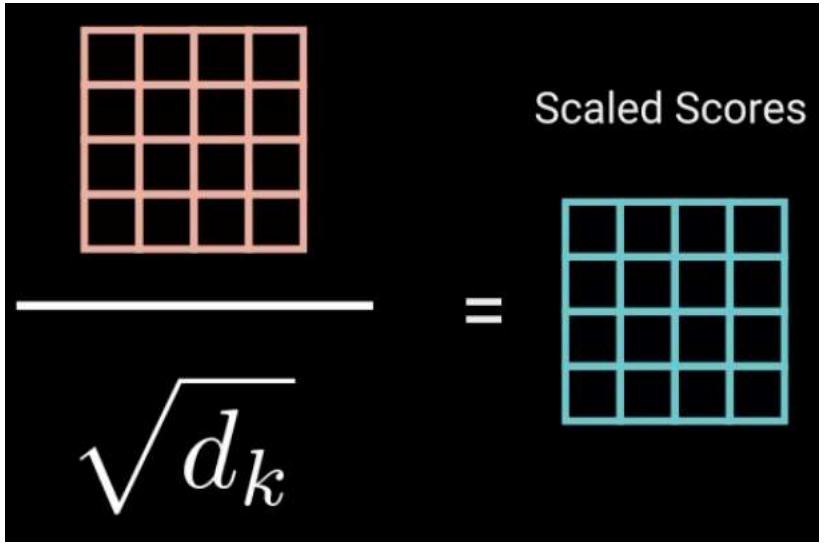


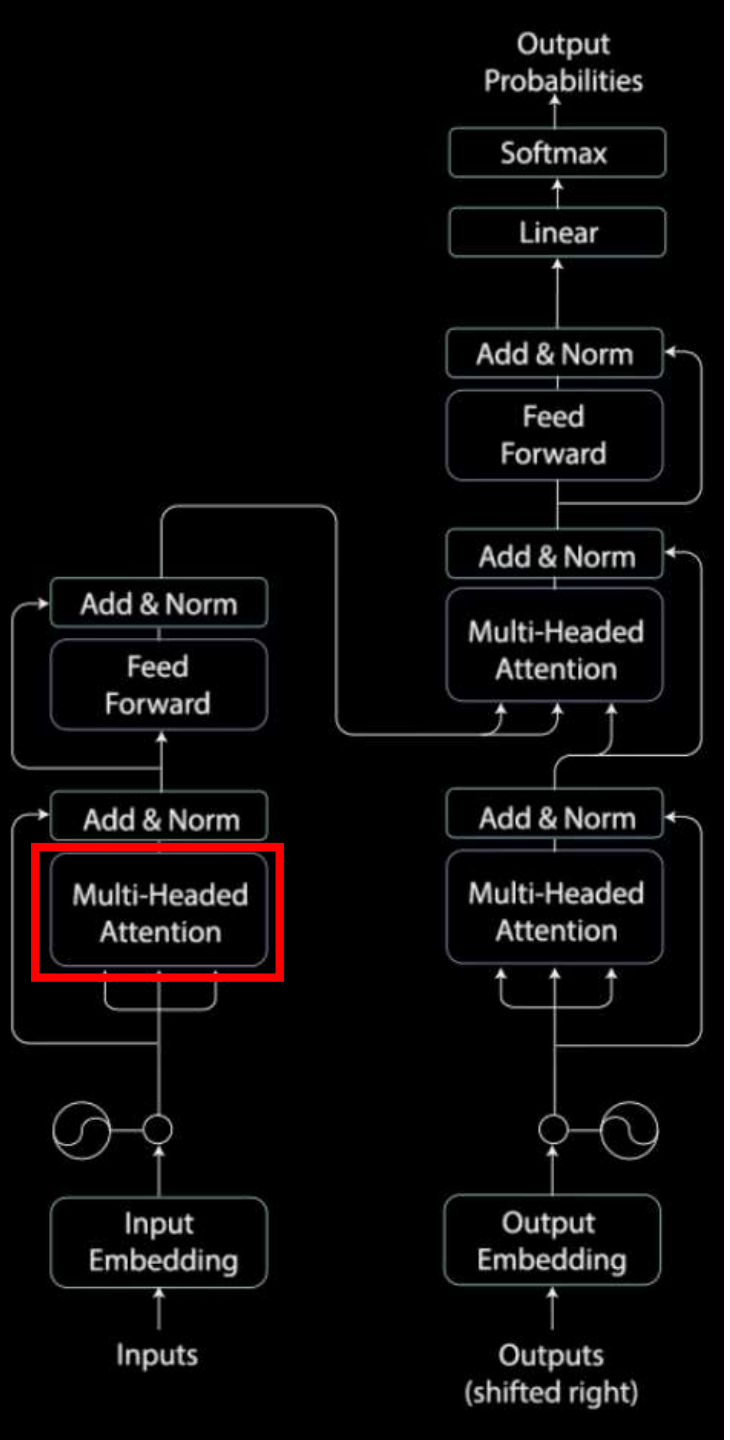


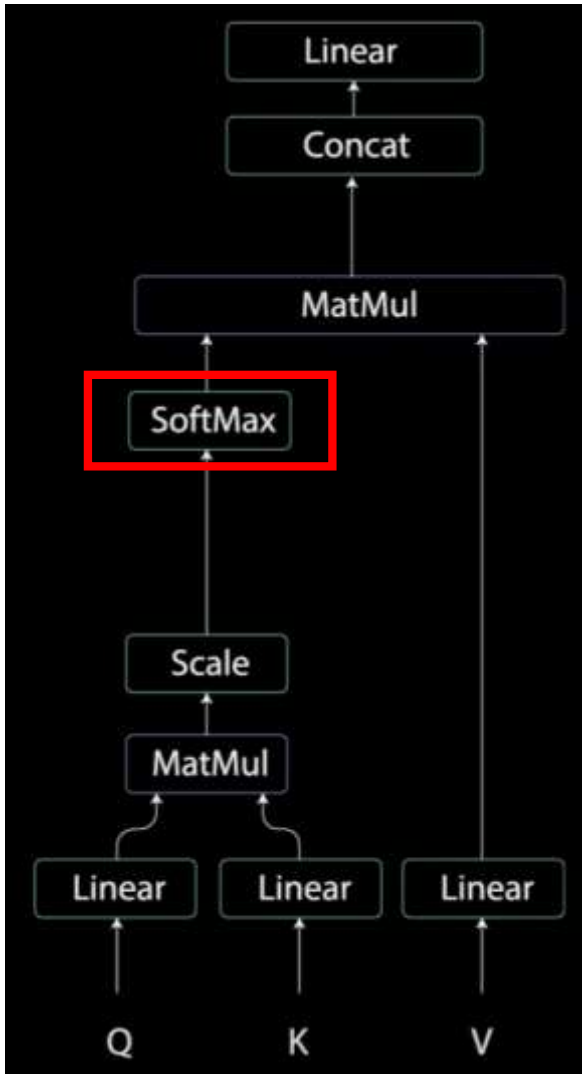
	Hi	how	are	you
Hi	98	27	10	12
how	27	89	31	67
are	10	31	91	54
you	12	67	54	92

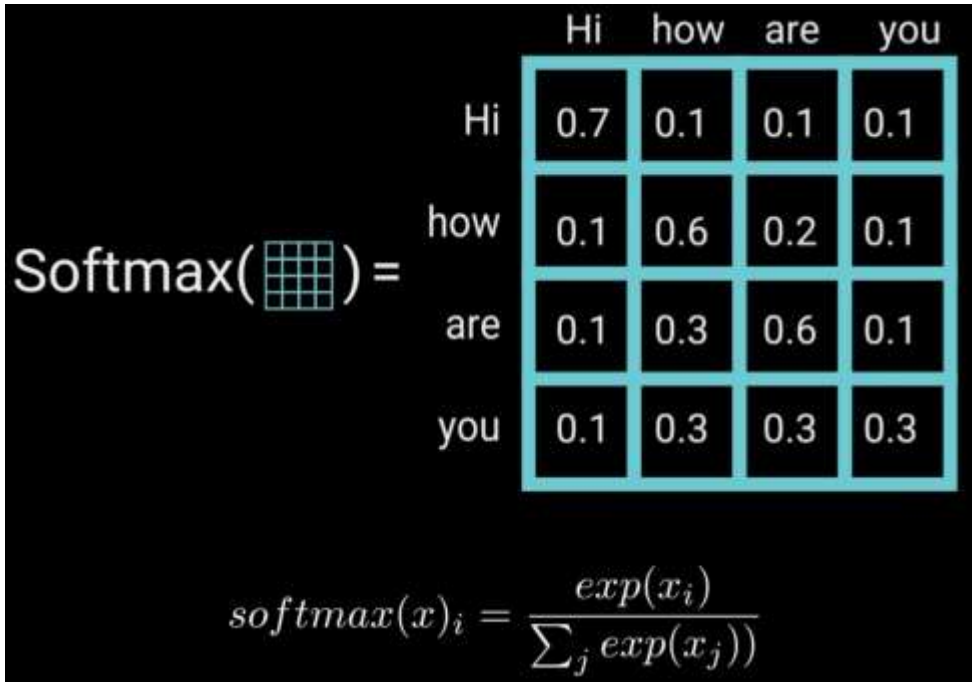

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Headed Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Headed Attention
Add & Norm
Multi-Headed Attention
Add & Norm
Multi-Headed Attention
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

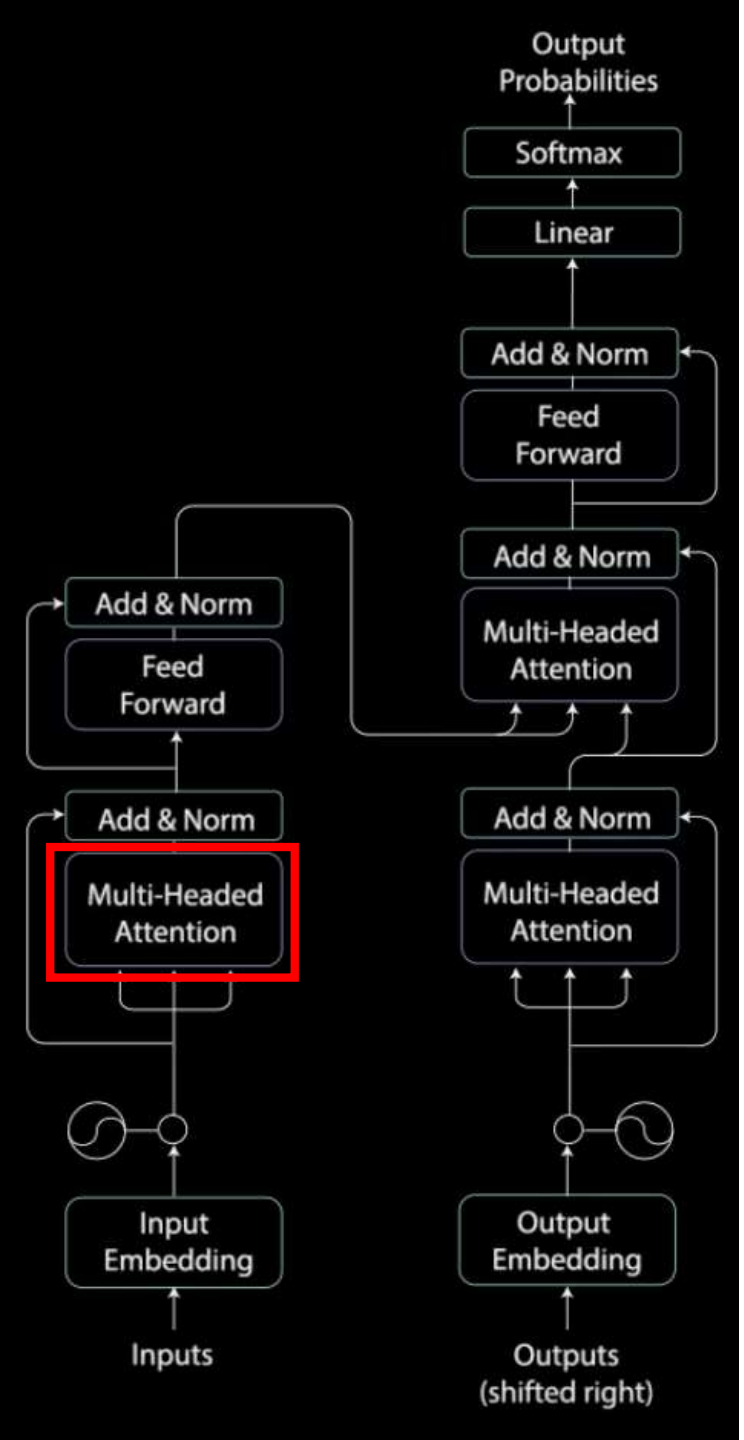

Linear
Concat
MatMul
SoftMax
Scale
MatMul
Linear
Linear
Linear
Q
K
V

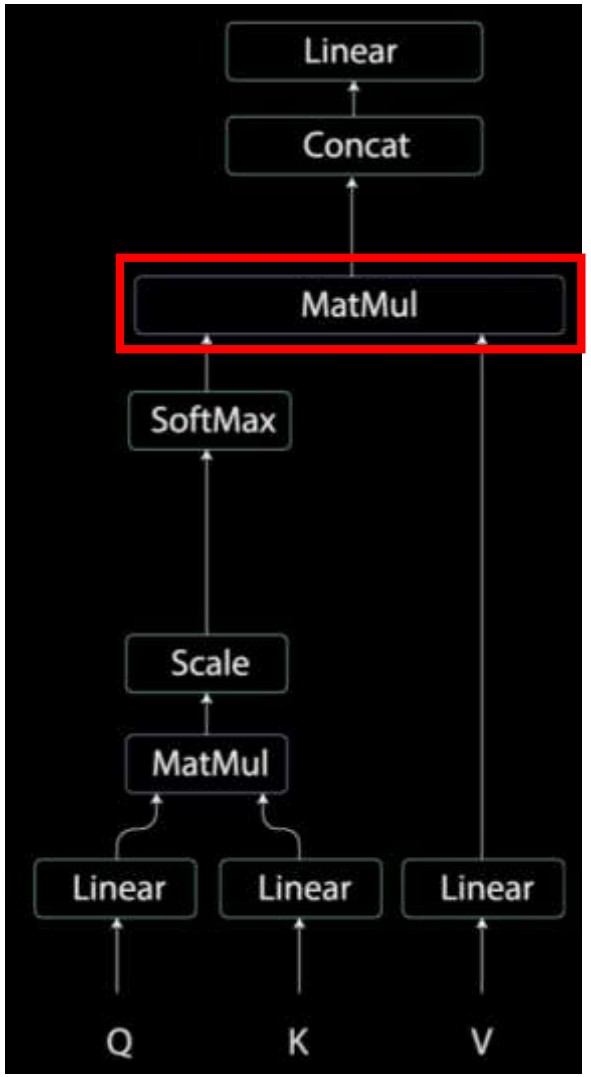

Scaled Scores
$\sqrt{d_k}$
=

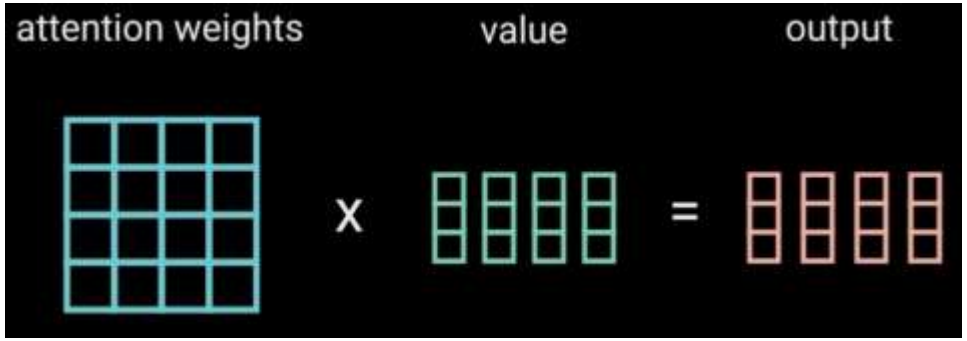

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Headed Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Headed Attention
Add & Norm
Multi-Headed Attention
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

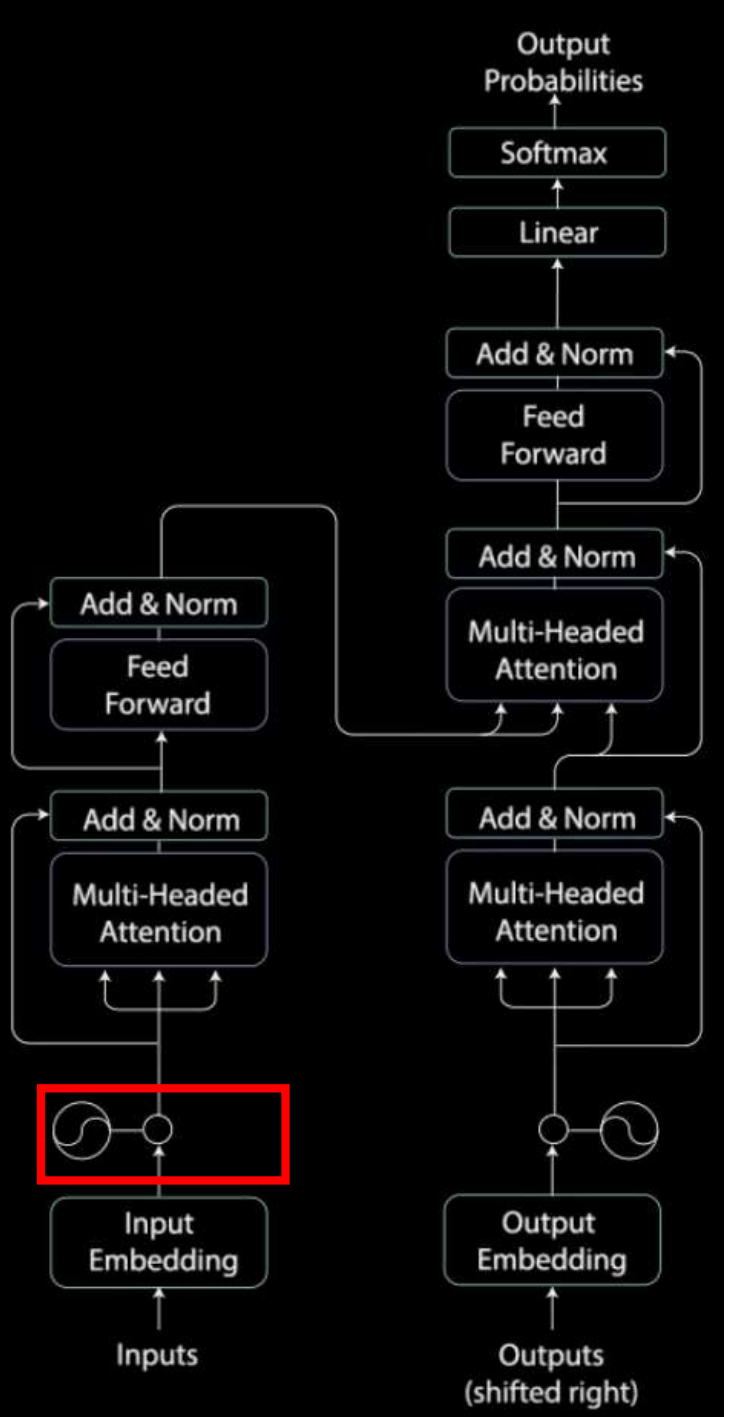

Linear
Concat
MatMul
SoftMax
Scale
MatMul
Linear
Linear
Linear
Q
K
V

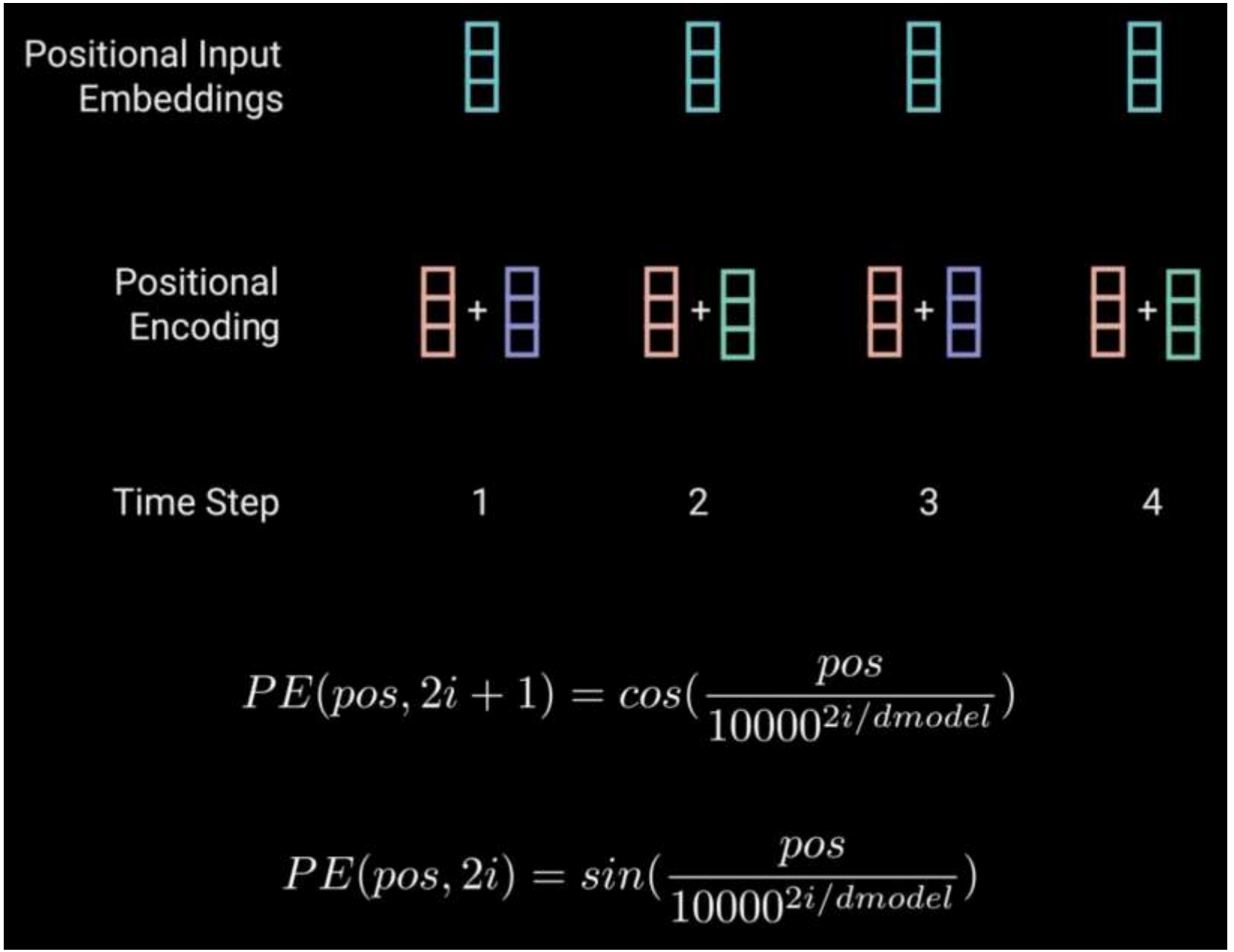

Softmax( ) =
Hi how are you
Hi 0.7 0.1 0.1 0.1
how 0.1 0.6 0.2 0.1
are 0.1 0.3 0.6 0.1
you 0.1 0.3 0.3 0.3
$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))}$


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Headed Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Headed Attention
Add & Norm
Multi-Headed Attention
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)


Linear
Concat
MatMul
SoftMax
Scale
MatMul
Linear
Linear
Linear
Q
K
V

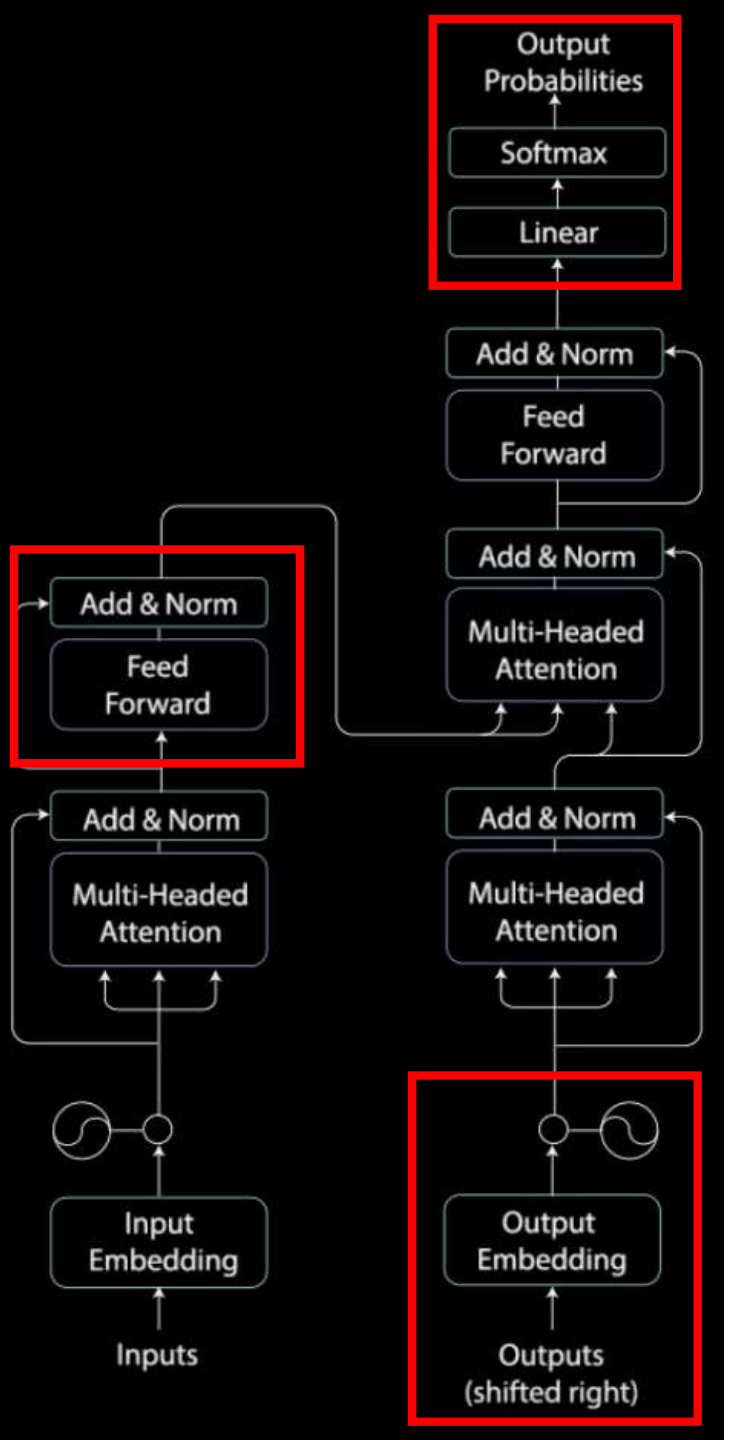

attention weights
value
output
X
=

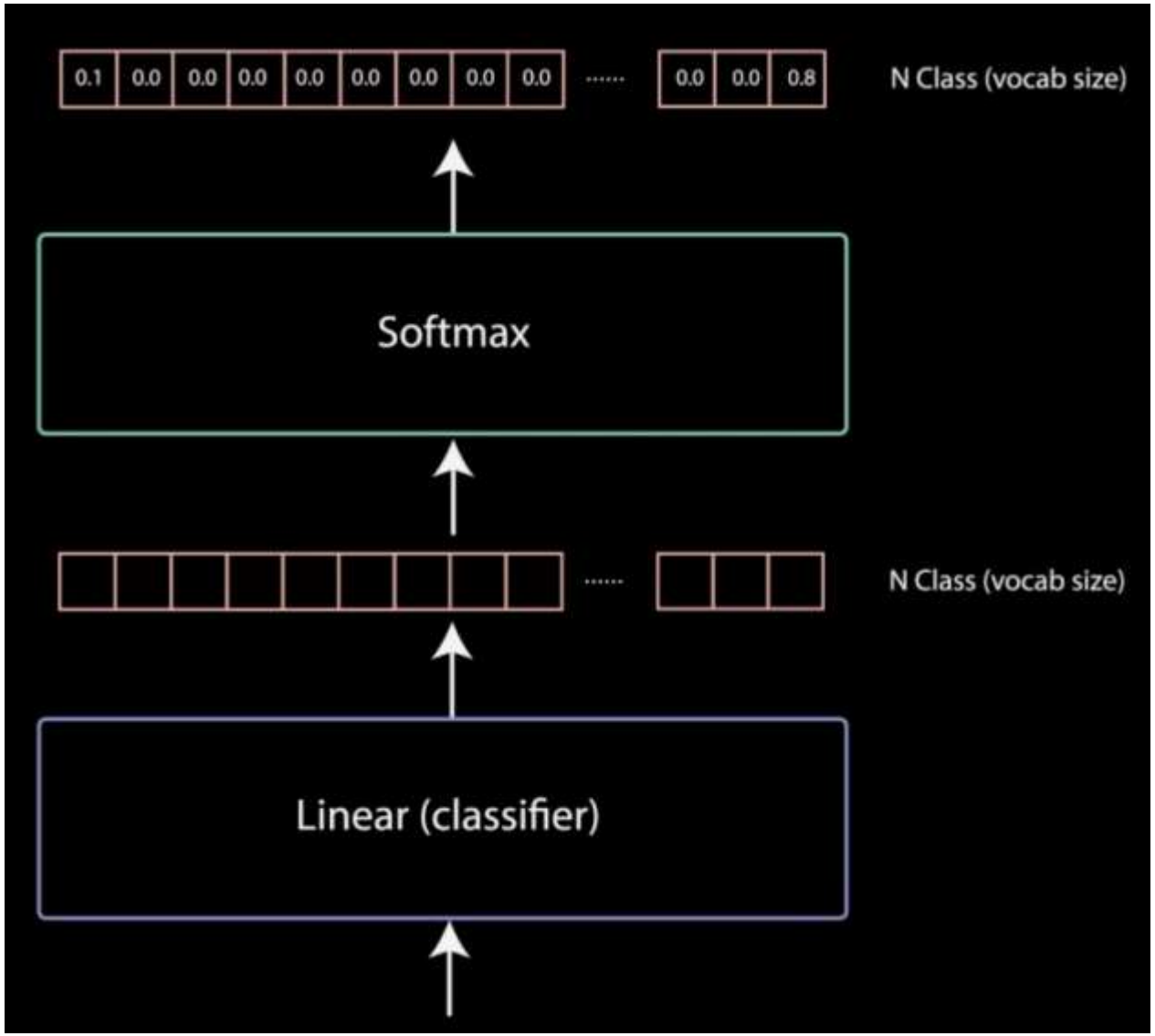




$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{2i/dmodel}})$$

$$PE(pos, 2i) = sin(\frac{pos}{10000^{2i/dmodel}})$$


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Headed Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Headed Attention
Add & Norm
Multi-Headed Attention
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

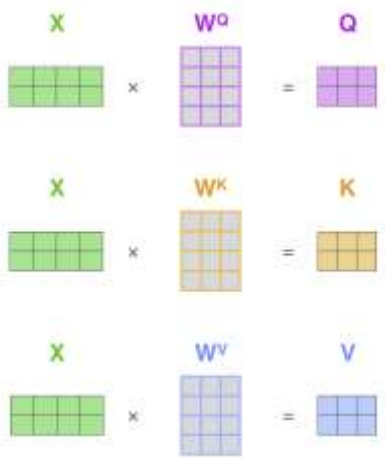

0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ..... 0.0 0.0 0.8
N Class (vocab size)
Softmax
N Class (vocab size)
Linear (classifier)

Queries, keys, and values

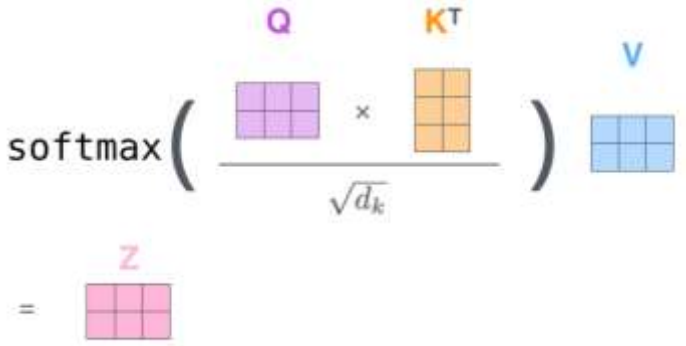
The Transformer paper redefines the attention mechanism by providing a generic definition based on ***queries, keys, values.***

Intuition: Use the **query** of the target and the **key** of the input to calculate a matching score.

These [illegible] scores act as the weights of the **va**[illegible].



The input consists of queries and keys of dimension d_k, and values of dimension d_v.



Compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values.

Attention: dot product with scaling

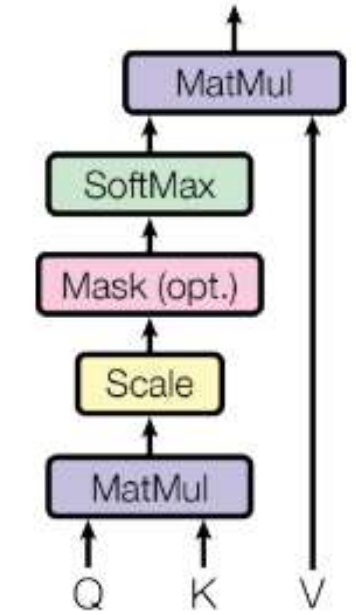
Attention mechanism in transformers are "scaled dot-product attention".

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

Dot product computes similarity between queries and keys.

For large values of d_k the dot products grow in magnitude, pushing the softmax function into with extremely small gradients.

The scaling by $\frac{1}{\sqrt{d_k}}$ counteracts this.



Multi-headed attention

$$MultiHead(Q,K,V) = Concat(head_1,\dots,head_h)W^O,$$

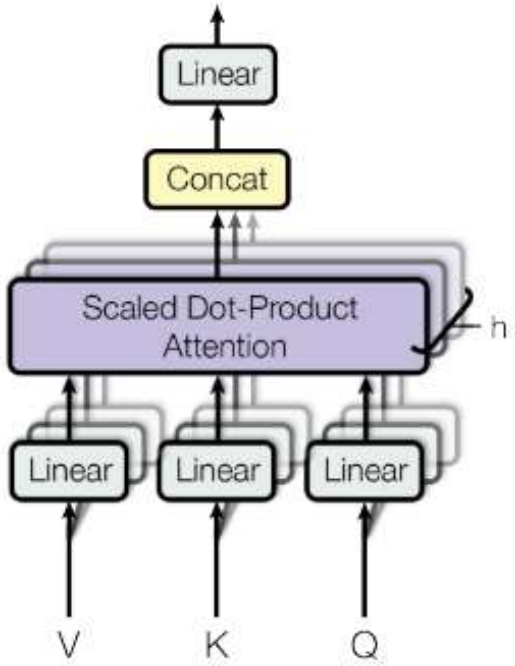
$$W^O{}_i \in \mathbb{R}^{hd_v \times d_{model}}.$$

It is beneficial to linearly project the Q, K and V, h times with different, linear projections to d_k, d_k and d_v dimensions.

The attention function is performed in parallel, on each of these projected versions of Q, K and V.

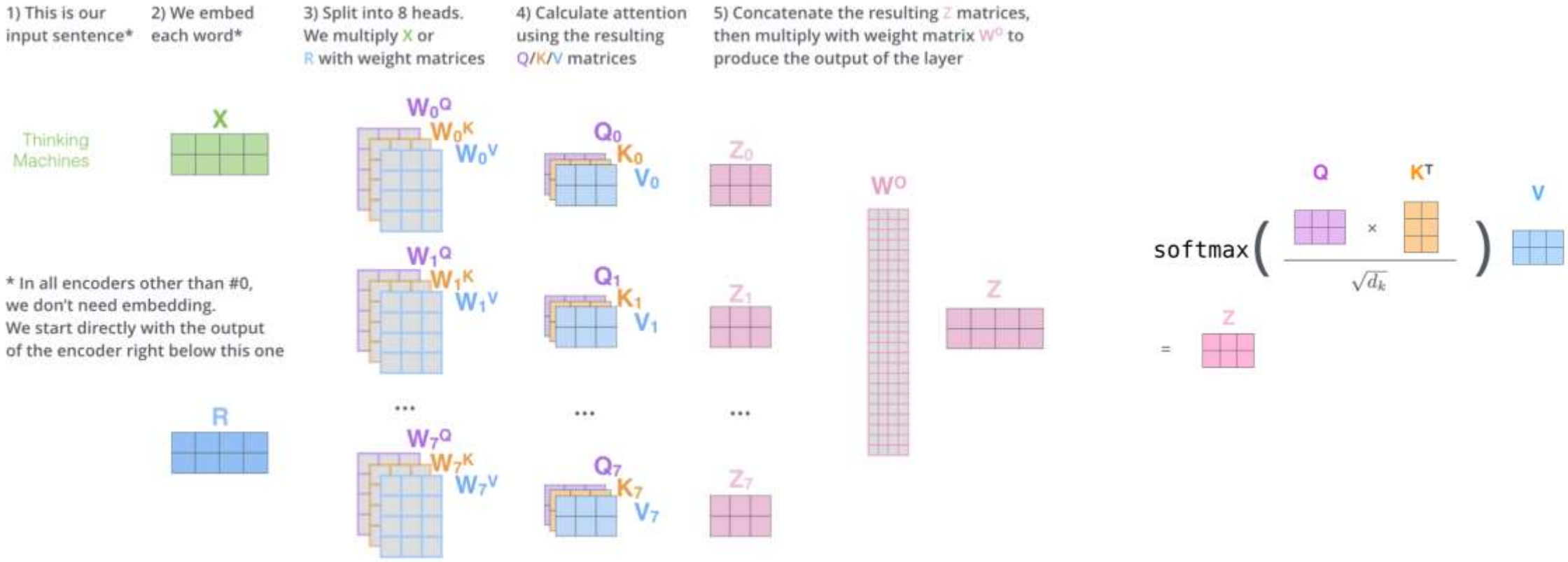
These are concatenated and again projected, resulting in the final values.

Similar to the multiple



Multi-headed self-attention

For multi-head self-attention: the ***queries***, ***keys***, ***values*** are equal to the input representation or from the previous (encoding/decoding) layer.



Break

Attention: dot product with scaling

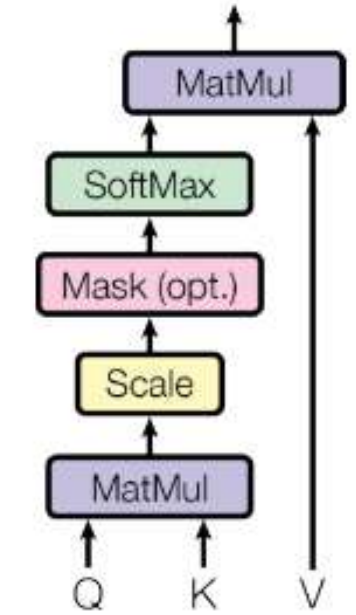
Attention mechanism in transformers are "scaled dot-product attention".

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

Dot product computes similarity between queries and keys.

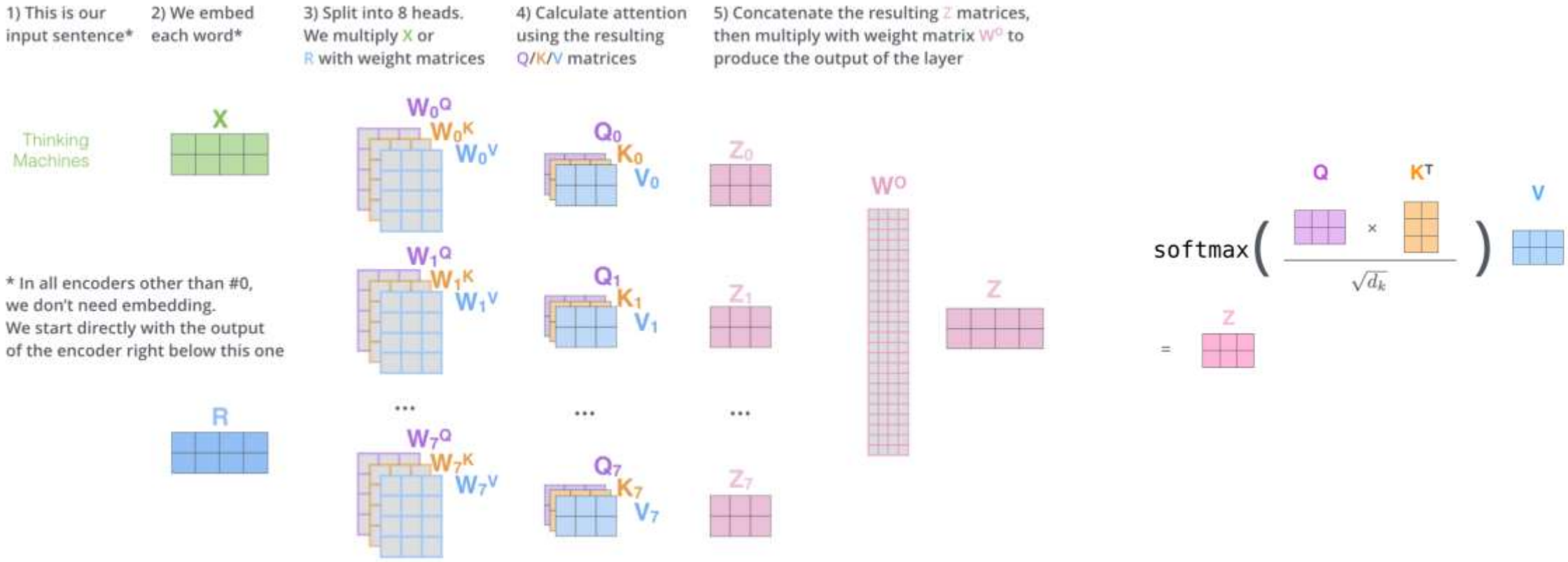
For large values of d_k the dot products grow in magnitude, pushing the softmax function into with extremely small gradients.

The scaling by $\frac{1}{\sqrt{d_k}}$ counteracts this.

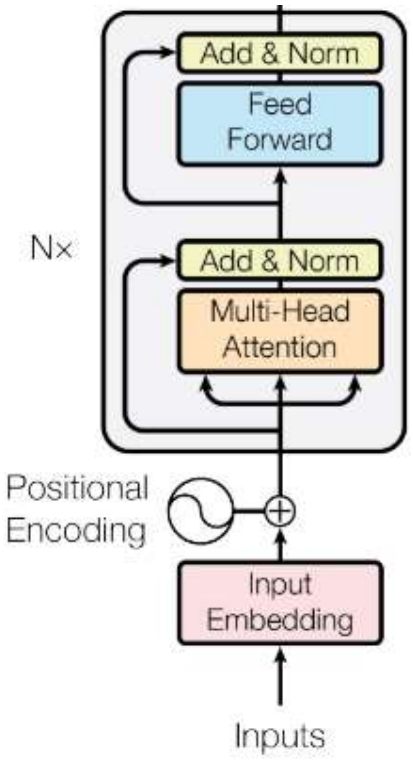


Multi-headed self-attention

For multi-head self-attention: the ***queries***, ***keys***, ***values*** are equal to the input representation or from the previous (encoding/decoding) layer.



Transformer encoder

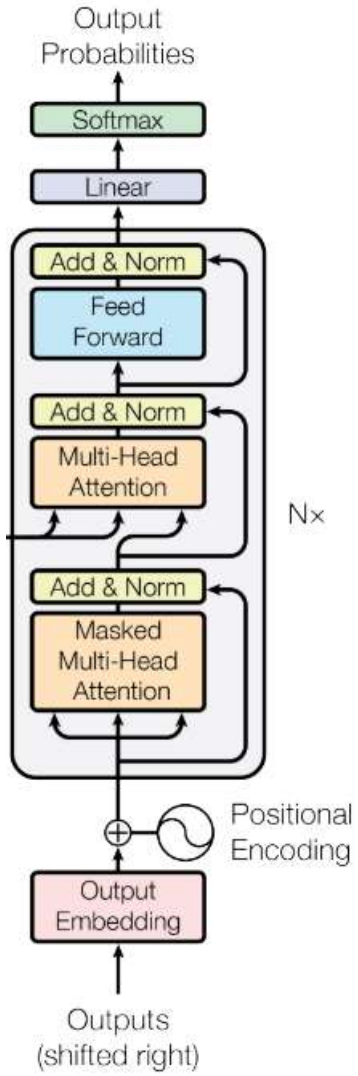


The encoder consists of N = 6 identical layers

Each encoder layer has 2 sub-layers: multi-head attention and fully connected feed-forward network.

Each sub-layer has a residual connection around it, followed by layer normalization.

Transformer decoder

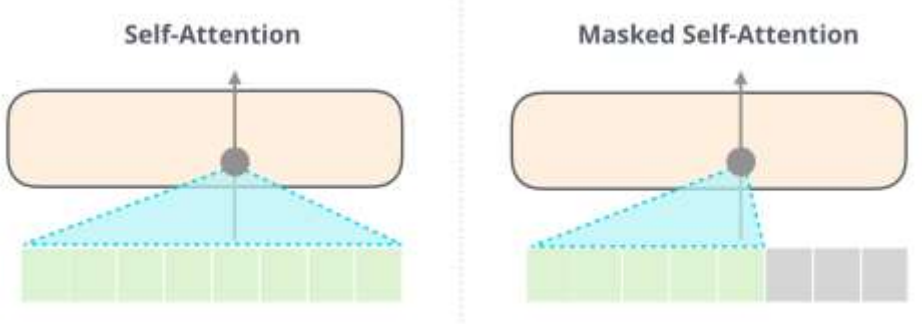


The decoder also consists of N = 6 identical layers

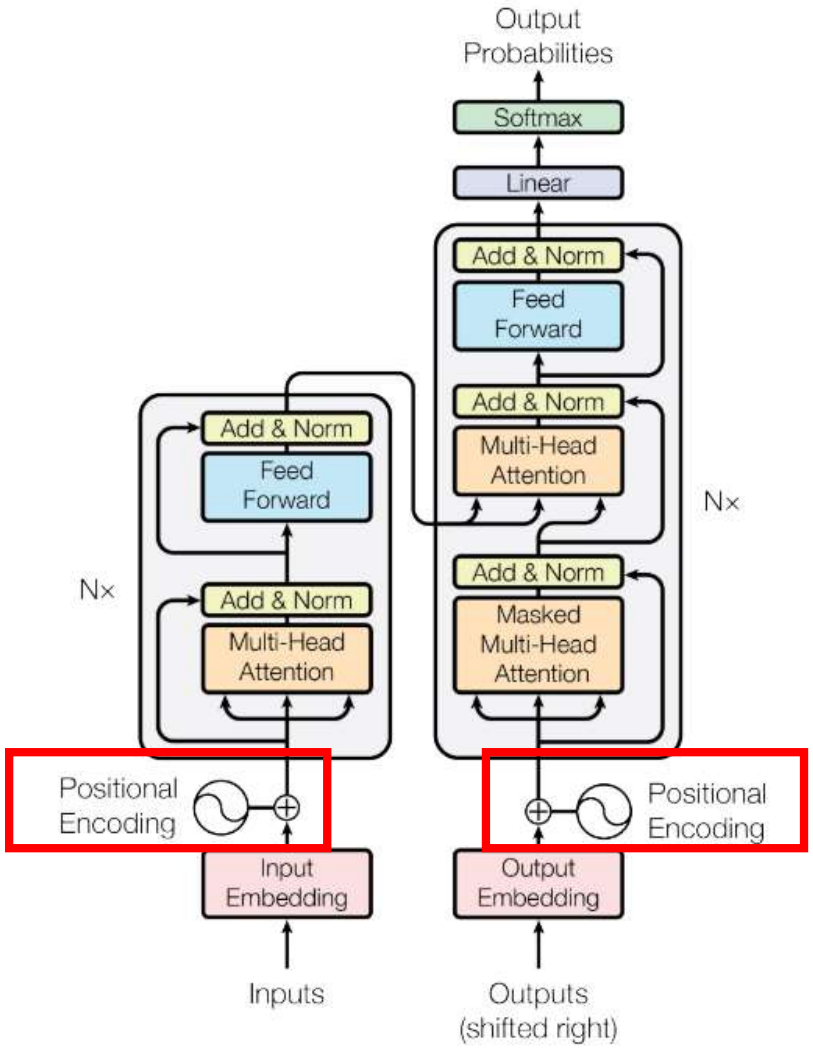
- A decoder layer is identical to the encoder layer,
- It has an additional 3rd sub-layer,
- Performs multi-head attention over the output of the encoder.

The masked self-attention sub-layer in the decoder prevents positions from attending to subsequent positions.

- the predictions for y on the known outputs



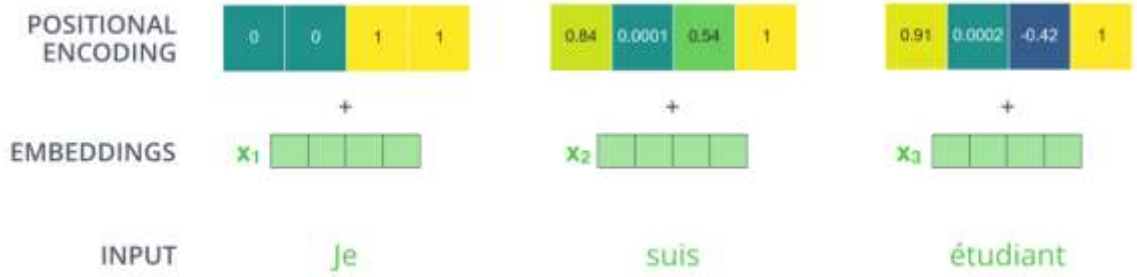
Position encoding



Attention is a permutation-invariant operation.

A pure attention module will return the same output regardless of the order of its inputs.

Solution: Positional encodings are added to the input in order to make use of the order of the sequence.



Positional encoding in the transformer

Intuitively: Positional encodings follow a specific pattern that the model learns

To determine the position of each word / the distance between words in the sequence.

Encode spatial, temporal, and modality identity… they can be learned or fixed.

The original Transformer uses sine and cosine functions of different frequencies:

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

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

Positional encoding in code

```
class PositionalEncoding(nn.Module):
    def __init__(self, d_model: int, dropout: float = 0.1, max_len: int = 5000):
            self.dropout = nn.Dropout(p=dropout)
            position = torch.arange(max_len).unsqueeze(1)
            div_term = torch.exp(torch.arange(0, d_model, 2) * (-math.log(10000.0) / d_model))
            pe = torch.zeros(max_len, 1, d_model)
            pe[:, 0, 0::2] = torch.sin(position * div_term)
            pe[:, 0, 1::2] = torch.cos(position * div_term)

    def forward(self, x: Tensor) -> Tensor:
    """ Args: x: Tensor, shape [seq_len, batch_size, embedding_dim] """
            x = x + self.pe[:x.size(0)]
            return self.dropout(x)
```

Pros:

- Transformer operates on data in parallel which accelerates the learning process, compared to RNN which operates sequentially
- Transformer can deal with long-term

Cons:

- Transformer scales quadratically with the number of inputs
- They are memory-intense and require lots of data and long training

Summary

Encoder-decoder is a useful architecture formany deep learning problems

Traditional encoder-decoder for NMT has the issue with the context vector

Attention mechanism overcomes the problem, by learning to select important features

Transformer is the first model that entirely relies on attention.

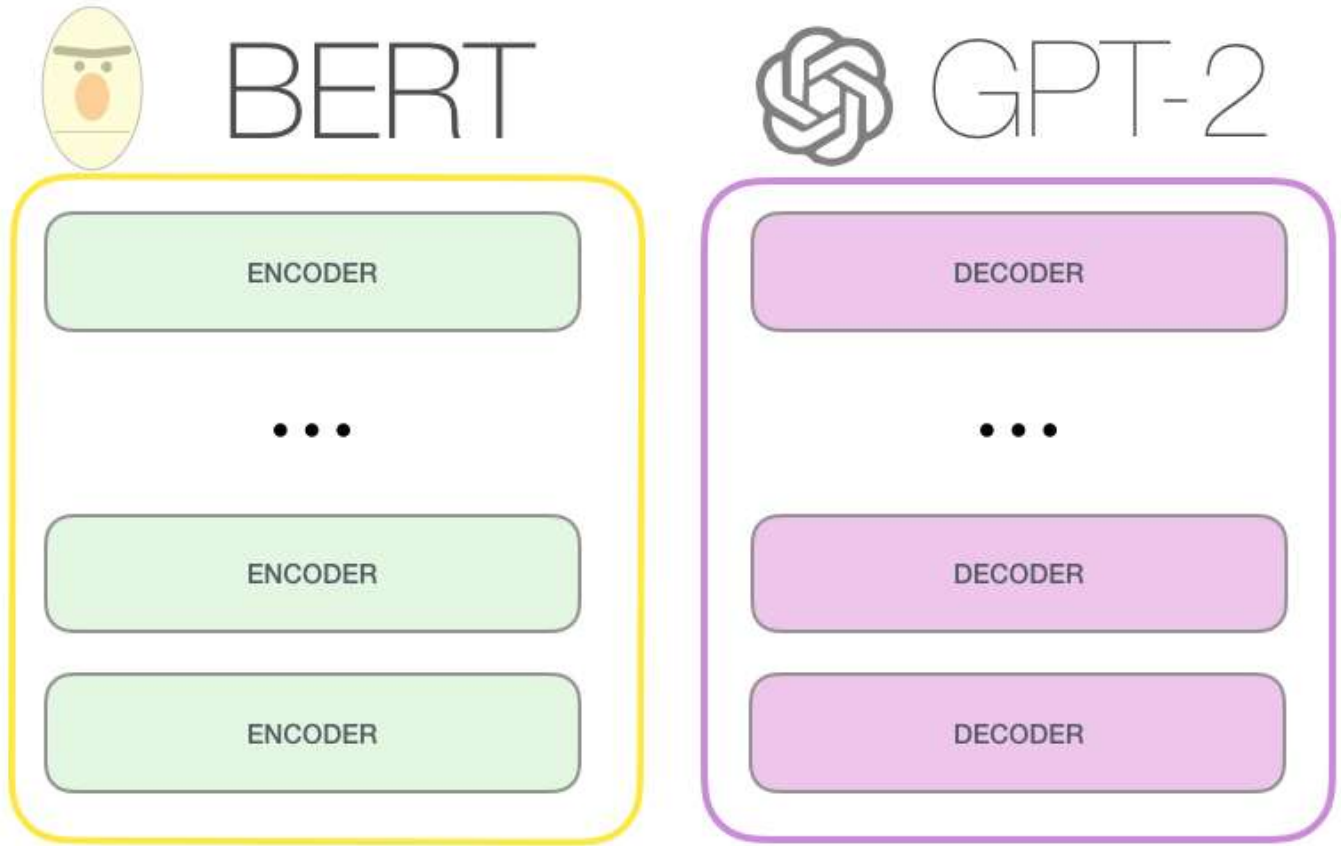
Recommended papers that started it all

Neural Machine Translation by Learning to Align and Translate, Bahdanu et al. ICLR (2015)

Long Short-Term Memory-Networks for Machine Reading, Cheng et al. (2016)

Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, Xu et al. (2016)

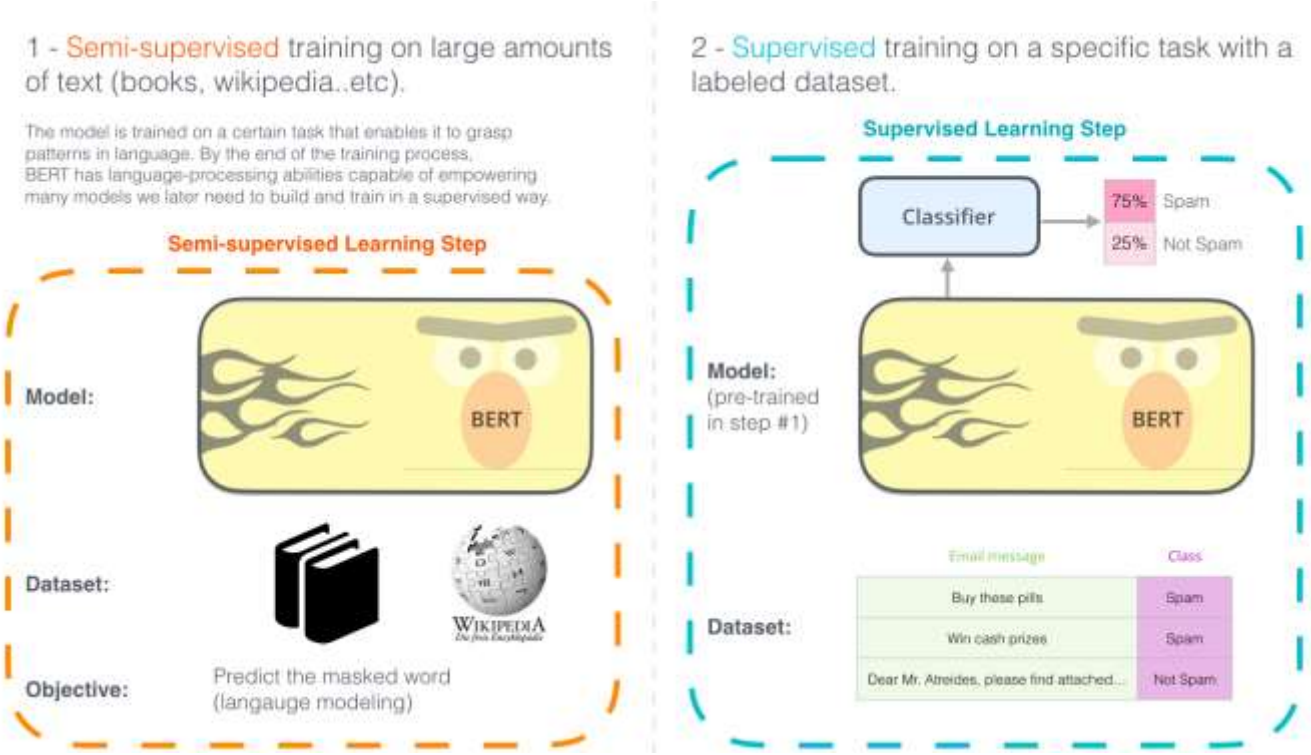
Attention Is All You Need, Vaswani et al. (2017)


BERT
GPT-2
ENCODER
ENCODER
ENCODER
DECODER
DECODER
DECODER

BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers

Idea: pre-train bidirectional representations from unlabeled text, by jointly conditioning on both left and right context.

A pre-trained BERT model can be fine-tuned with just one additional output layer to create SOTA models



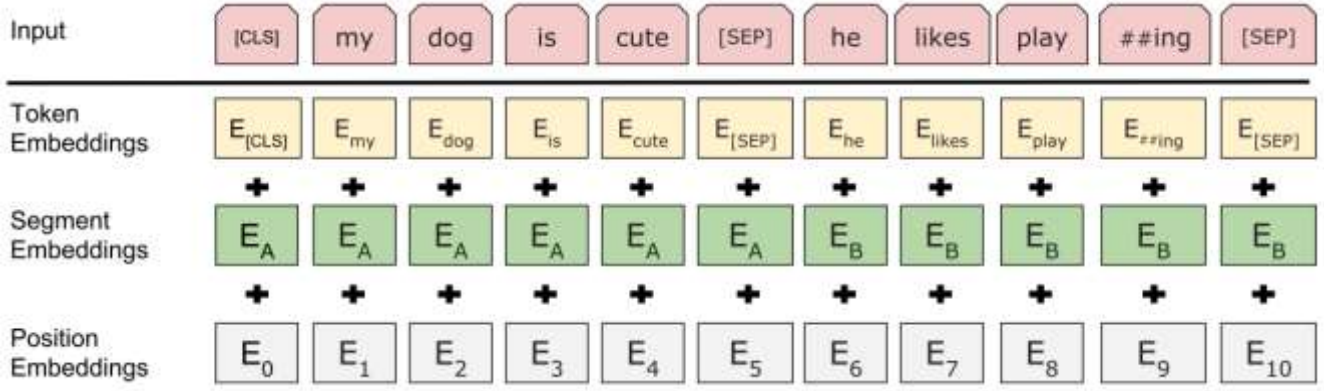
BERT input representation

The input representation: single sentence OR a pair of sentences in one sequence.

The complete input is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

The first token of every sequence is always a special classification token ([CLS]).

- The final hidden state corresponding to this token is used as the aggregate sequence representation for



Sentence pairs are packed together into a single sequence, separated with a special token ([SEP]).

Also, a learned embedding is added to every token indicating whether it belongs

BERT pre-training

Task #1: **M**asked **L**anguage **M**odelling (MLM)

- Mask a percentage of the input tokens at random with a special token [MASK], and then predict those masked tokens.

Task #2: **N**ext **S**entence **P**rediction (NSP)

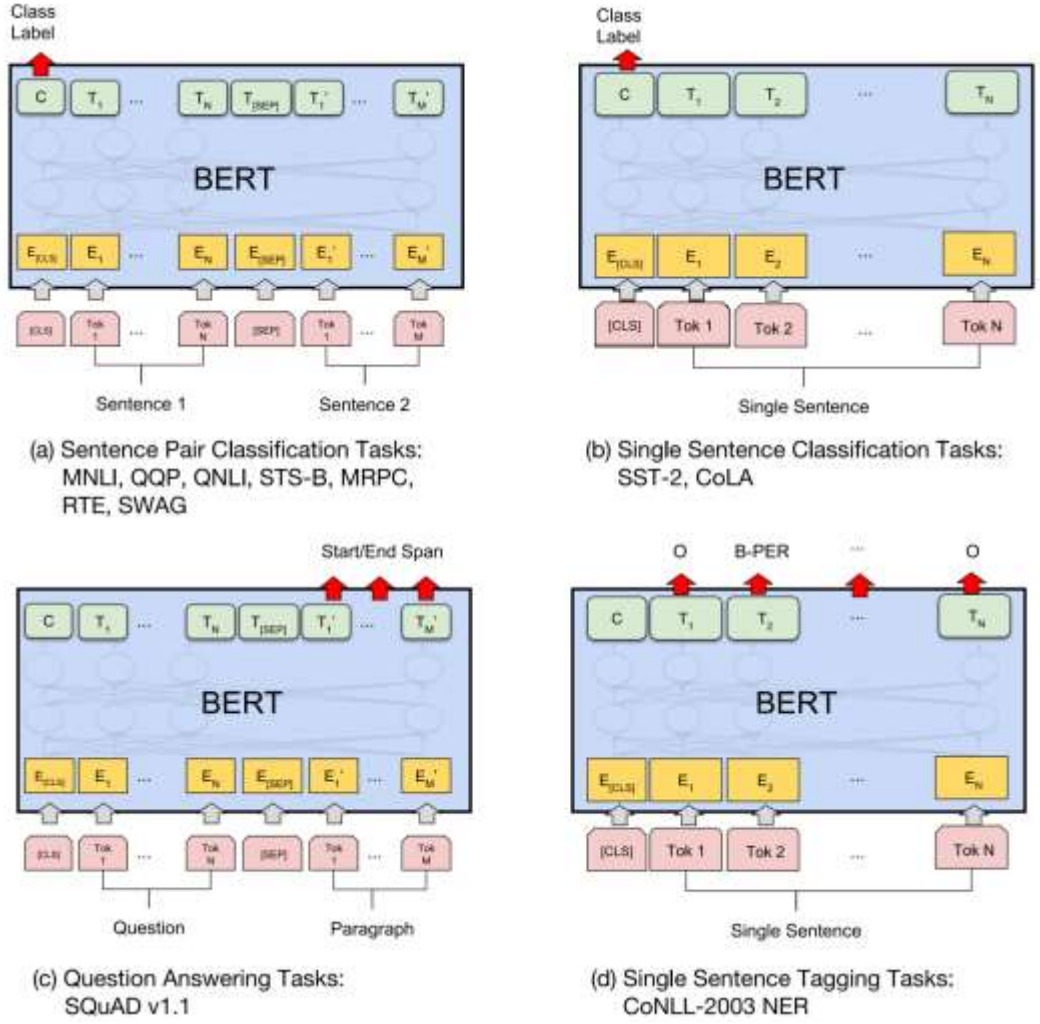
- Binary prediction whether the next sentences is a correct one.
- When choosing the sentences A and B, 50% of the time B is the actual next sentence that follows A and 50% of the time it is a random sentence from the corpus.

Datasets: BooksCorpus (800M words) and Wikipedia (2,500M words)

BERT fine-tuning

Fine-tuning is straightforward since the self-attention mechanism allows BERT to model many downstream tasks.

It is relatively inexpensive compared to pre-training.



A family of BERT models

RoBERTa: A Robustly Optimized BERT Pretraining Approach,

- More data, Longer training, Larger batches

ALBERT: A Lite BERT for Self-supervised Learning of Language Representations

DeBERTa

DistilBERT

CamamBERT

RoBERT

ClinicalBERT

GPT

Generative **P**retraining by **T**ransformers == GPT[1]

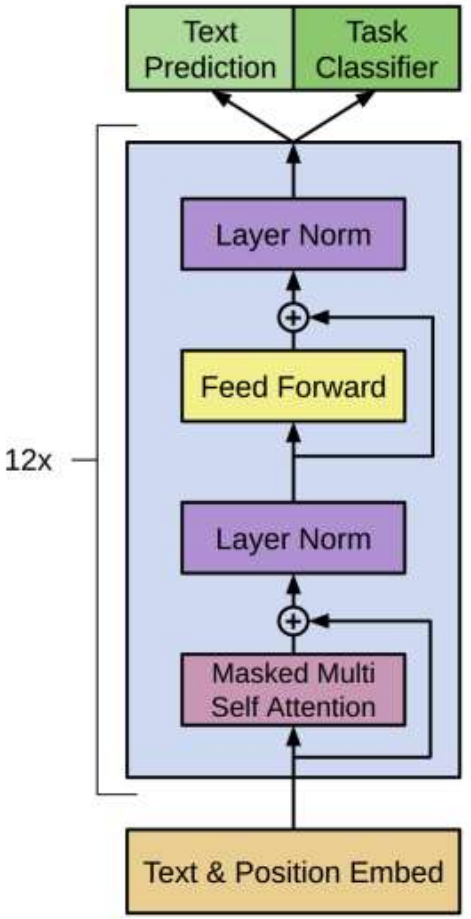
- A pre-trained unidirectional Transformer decoder

The idea: To train a generative language model using "unlabeled" data.

And then fine-tune it on specific downstream tasks.

Unsupervised pre-training:

- Given an unsupervised corpus of



GPT 1 to 3

GPT1[1] : Proves that language modeling serves as an effective pre-training objective which helps the model to generalize well

GPT2[2] : uses a larger dataset for training and adds additional parameters to build a stronger language model.

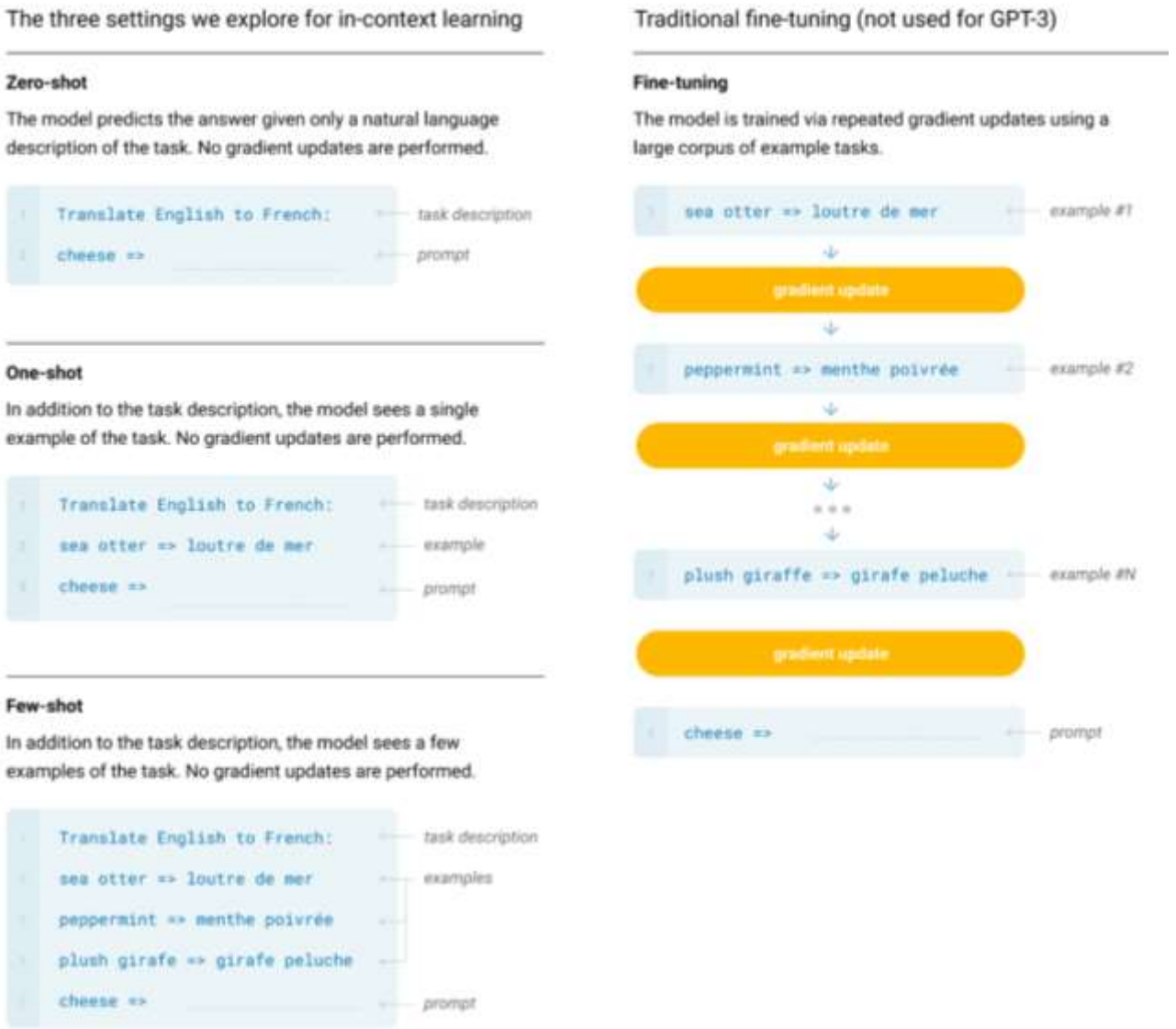
GPT3[3] : even larger than GPT2, can automatically generate high-quality paragraphs.

- Performs well on tasks on which it was never explicitly trained on, like writing SQL natural language description

	GPT-1	GPT-2	GPT-3
Parameters	***117 Million***	***1.5 Billion***	***175 Billion***
Decoder Layers	12	48	96
Hidden Layer	768	1600	12288
Batch Size	64	512	3.2M

[1] Improving Language Understanding by Generative Pre-Training, Radford et al. (2018)
[2] Language Models are Unsupervised Multitask Learners, Radford et al. (2019)
[3] Language Models are Few-Shot Learners, Radford et al. (2020)

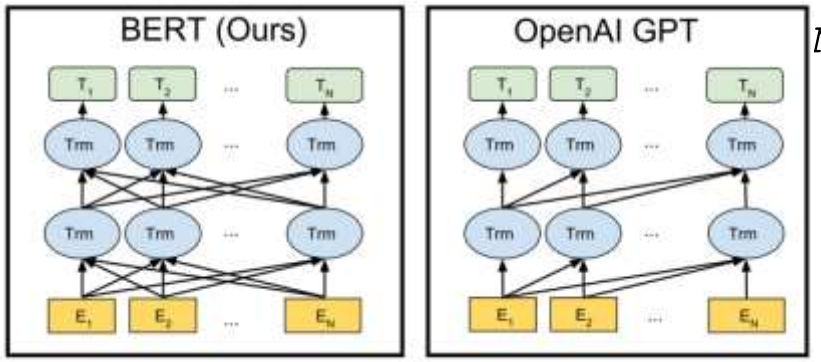
GPT enables in-context learning



GPT versus BERT

The key difference between the BERT and GPT, is that GPT is a unidirectional Transformer decoder, whereas BERT is bidirectional Transformer encoder.

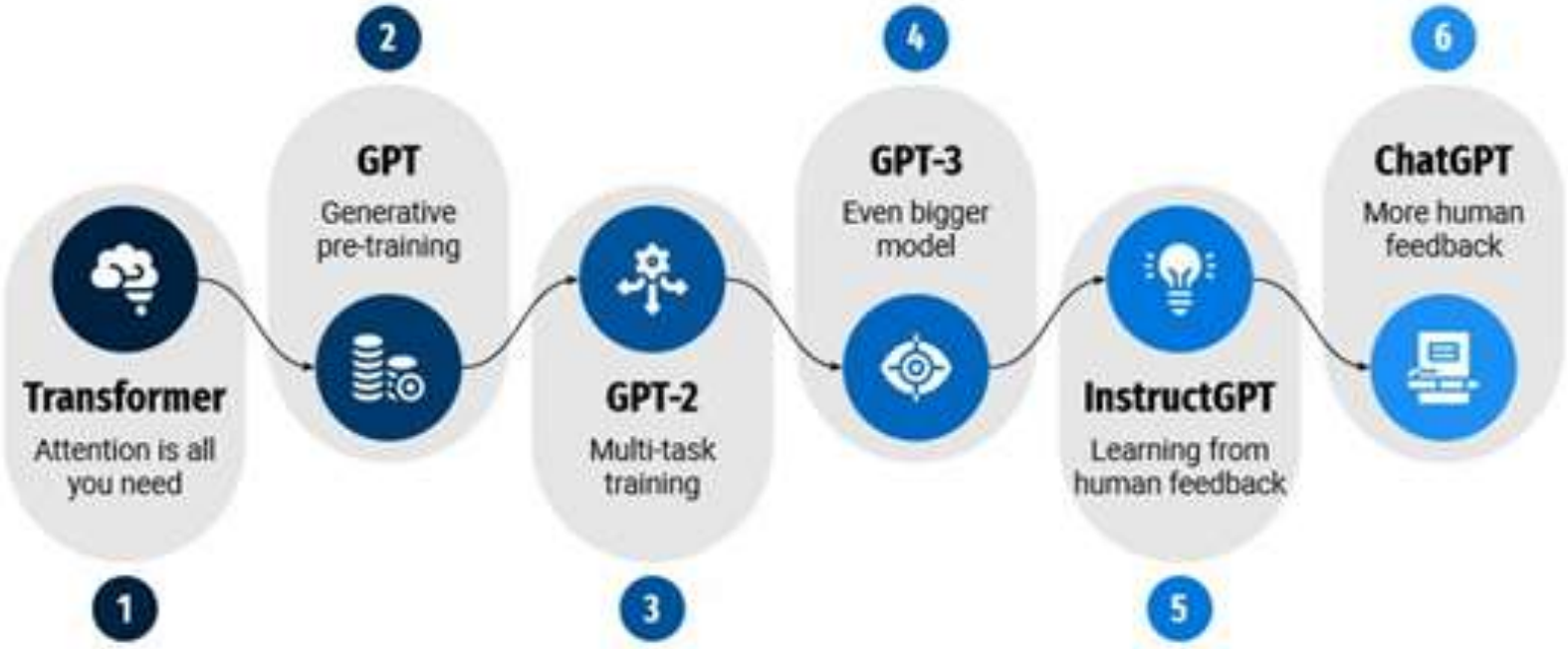
GPT outputs one token at a time, just like traditional language models.

- *After each token is produced, that token is added to the sequence of inputs.*
- *That new sequence becomes the input to the model in its next step.*
- *This is an idea* 

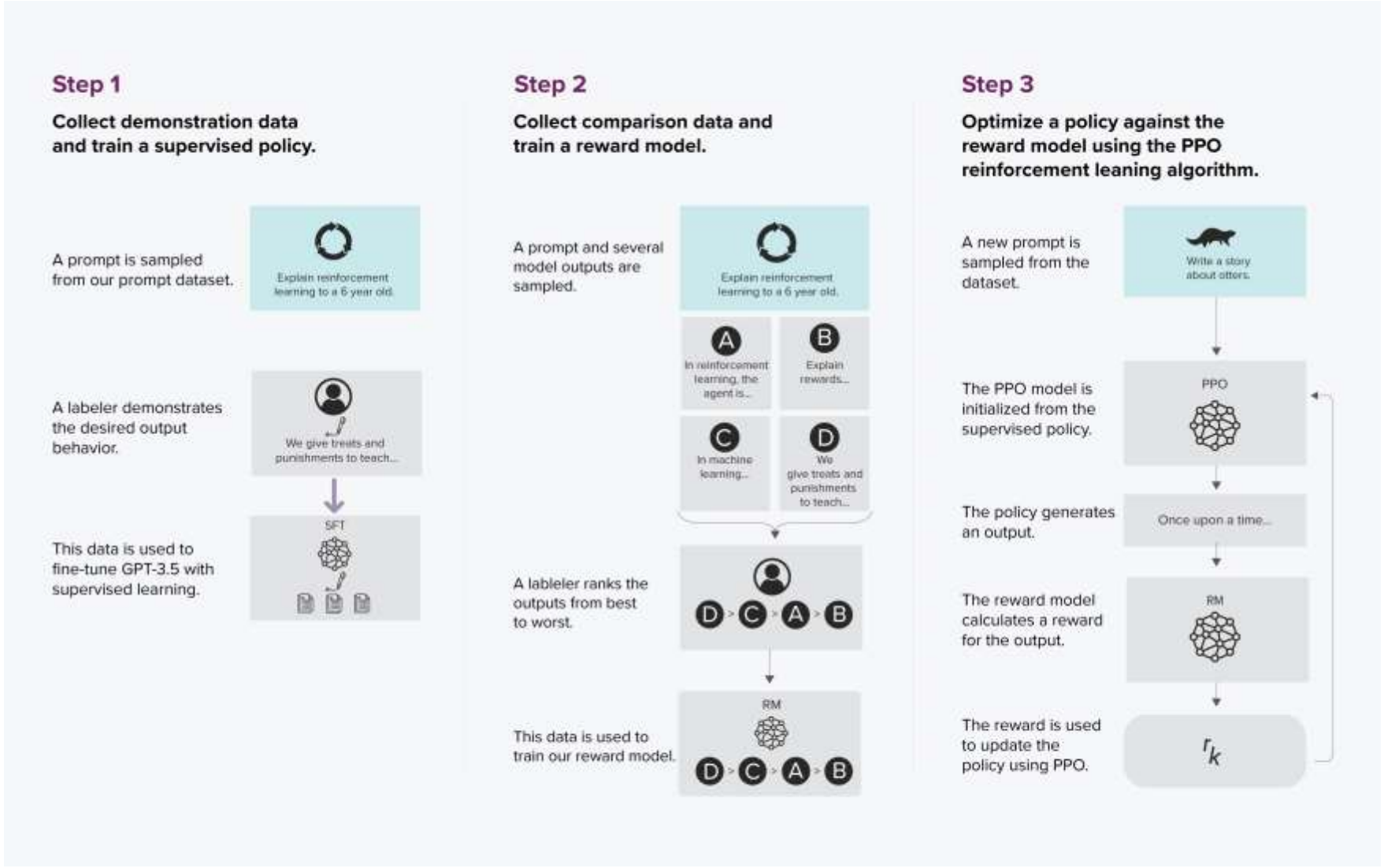
n".

In losing auto-reg the ability to incorporate the co f a word.

What is ChatGPT?



Reinforcement learning from human feedback



Why is ChatGPT so convincing

It is a result of exposure to extreme scale.

It’s network consists of billions of parameters.

It is a product of careful and elaborate hyperparameter tuning.

Human feedback RL creates output formats that we like.

In short, ChatGPT has condensed the internet and gives it to us in a pleasing narrvative style.

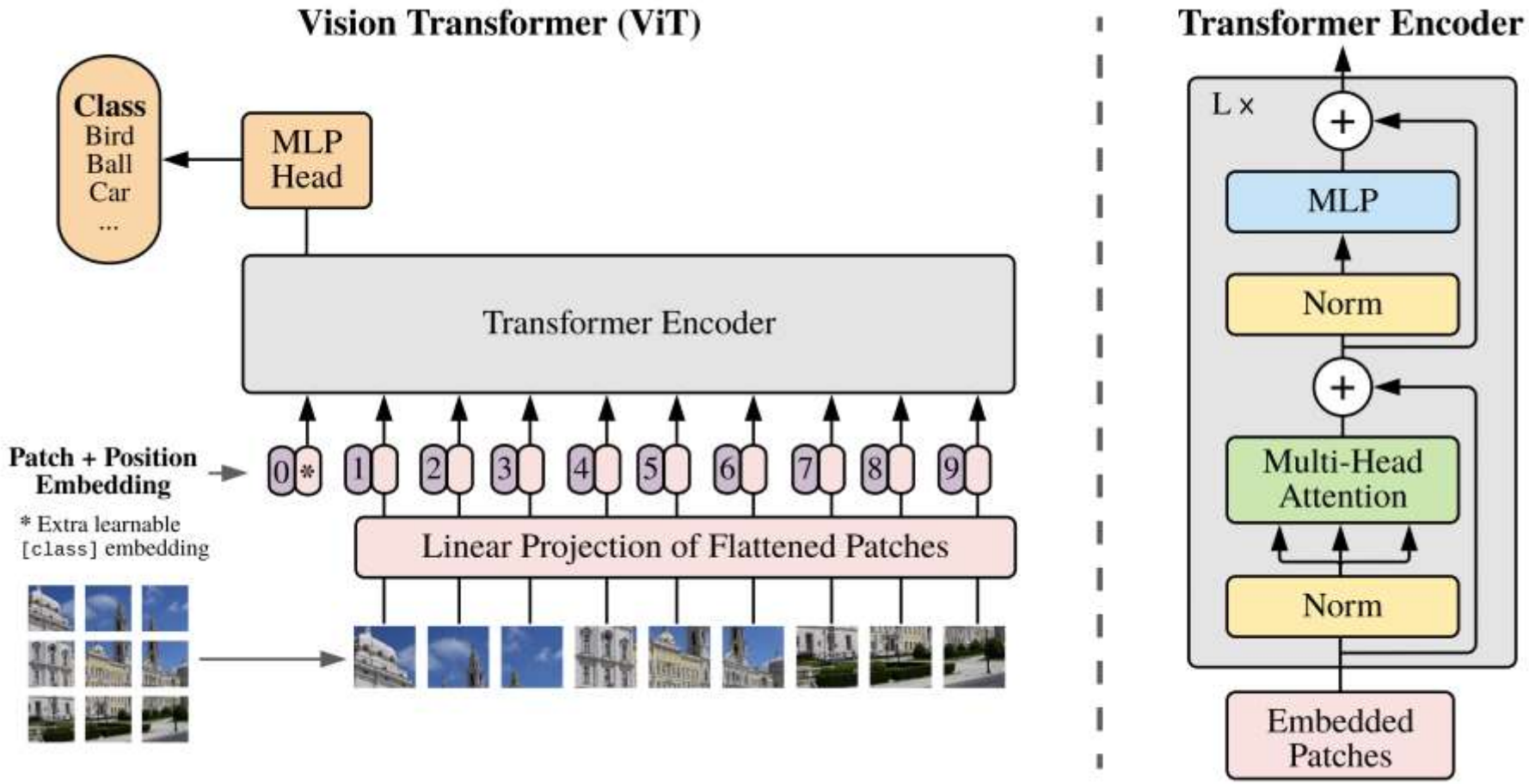
Vision transformers

Attention for text is intuitive, but it has also proven to be important for vision.

Transformers assume the input is a sequence of tokens.

What is a token in the context of an image?

Vision Transformer (ViT)



Vision Transformer (ViT)

Like BERT's [CLS] token, a learnable embedding is prepended to the sequence of embedded patches:

- the classification is done on this token (with an MLP)

"Position encodings" are added to the patch embeddings to retain positional information. (attention by itself doesn't have any notion of ordering/space)

- These vectors are also simply learned

```
# pos_embed has entry for class token, concat then add
if self.cls_token is not None:
    x = torch.cat((self.cls_token.expand(x.shape[0], -1, -1), x), dim=1)
x = x + self.pos_embed
```

Attention versus convolution

Convolution is local and shared.

Attention is global.

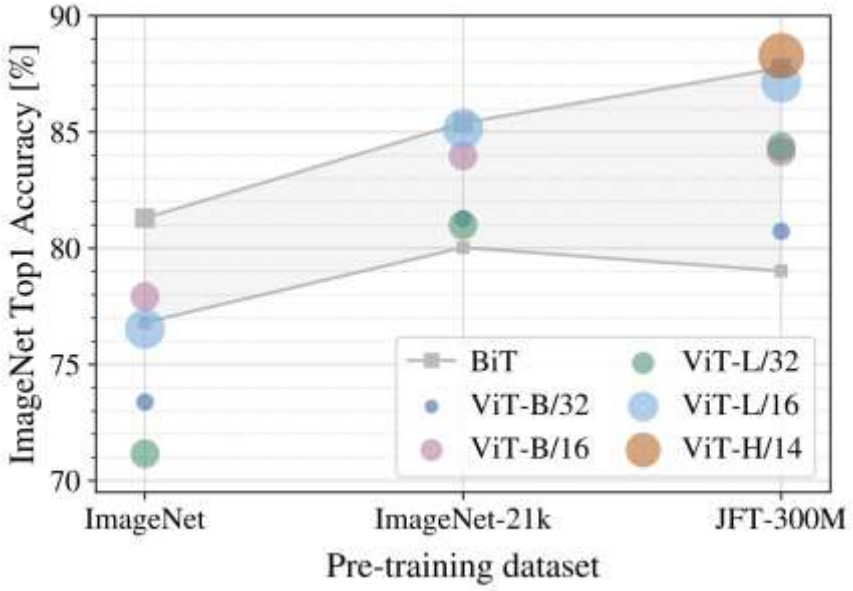
Hence both bring different views to visual representation learning.

Transformers impose less structure, useful in large-scale settings.

Training a ViT is more difficult

Original paper required pre-training onImageNet-22k (14M images) to achieve good performances.

DeiT (data efficient image transformers) paper showed training with ImageNet-1k possible if more augmentations and regularisations are used.



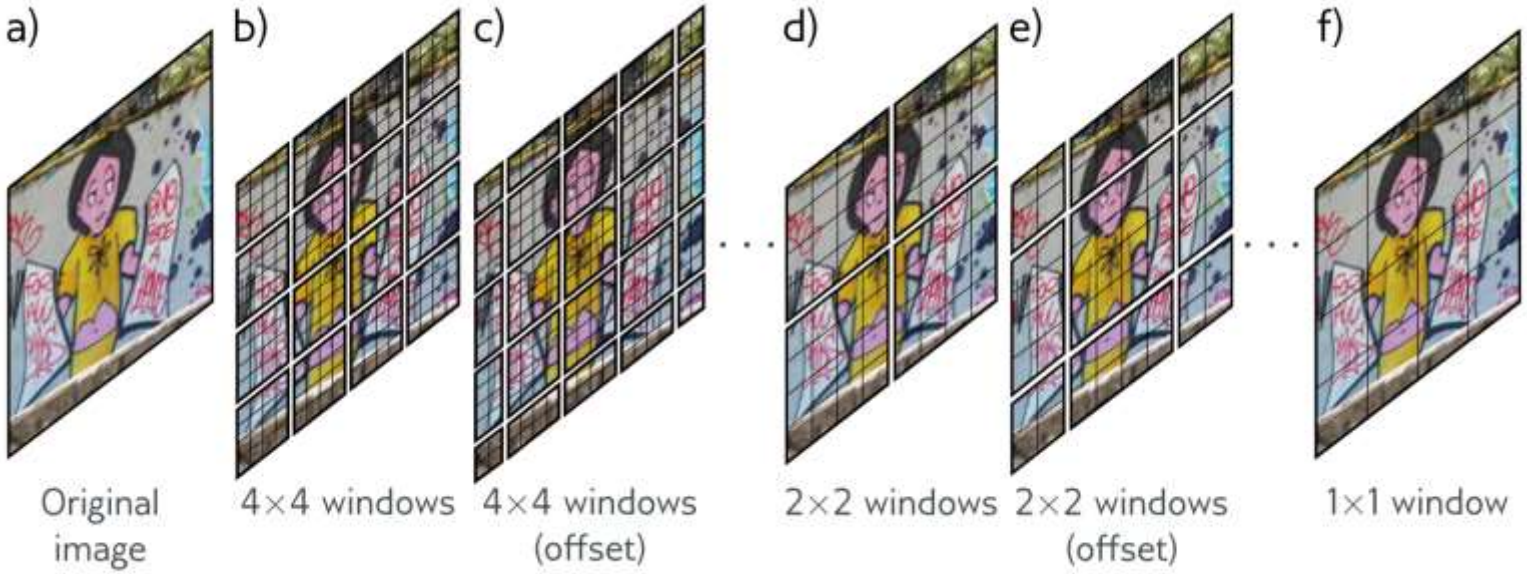
Ablation on ↓	Pre-training	Fine-tuning	Rand-Augment	AutoAug	Mixup	CutMix	Erasing	Stoch. Depth	Repeated Aug.	Dropout	Exp. Moving Avg.	top-1 accuracy: pre-trained 224^2	top-1 accuracy: fine-tuned 384^2
none: DeiT-B	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8 ±0.2	83.1 ±0.1
optimizer	SGD	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	74.5	77.3
	adamw	SGD	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8	83.1
data augmentation	adamw	adamw	✗	✗	✓	✓	✓	✓	✓	✗	✗	79.6	80.4
	adamw	adamw	✗	✓	✓	✓	✓	✓	✓	✗	✗	81.2	81.9
	adamw	adamw	✓	✗	✗	✓	✓	✓	✓	✗	✗	78.7	79.8
	adamw	adamw	✓	✗	✓	✗	✓	✓	✓	✗	✗	80.0	80.6
	adamw	adamw	✓	✗	✗	✗	✓	✓	✓	✗	✗	75.8	76.7
regularization	adamw	adamw	✓	✗	✓	✓	✗	✓	✓	✗	✗	4.3*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✗	✓	✗	✗	3.4*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✗	✗	✗	76.5	77.4
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✓	✗	81.3	83.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✓	81.9	83.1

Training data-efficient image transformers & distillation through attentio

Swin Transformer: return of locality

Idea: mostly looking at “local” neighborhood, so can save some computation (remember attention is O(n^2)) or gain some accuracy by modelling this.

Strong performance but slow models.



Convolutions and attention are complimentary

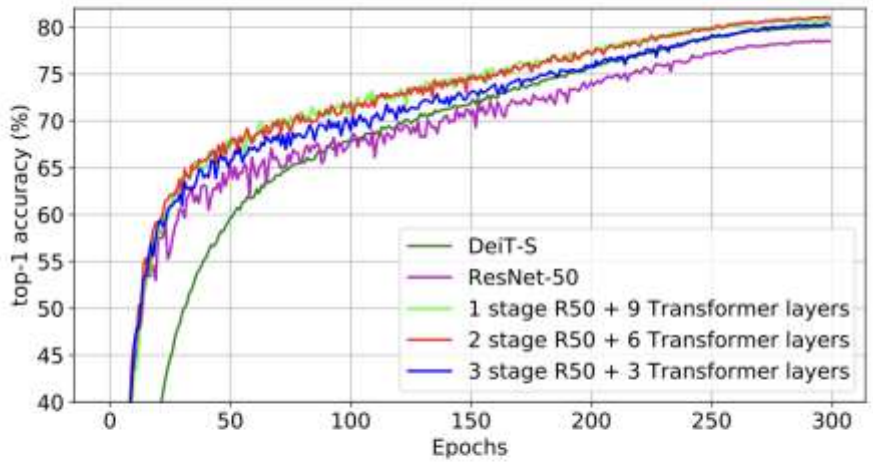
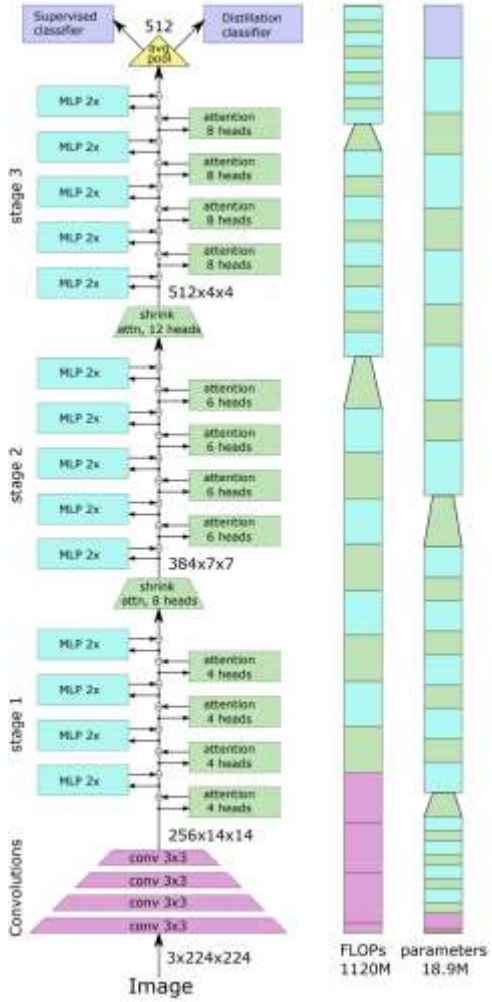


Figure 3: Models with convolutional layers show a faster convergence in the early stages compared to their DeiT counterpart.



LeViT: A Vision Transformer in ConvNet's Clothing for Faster Inference

Summary

Sequential modelling

Attention and self-attention

Transformers

Language and vision transformers

Next lecture

Lecture	Title
1	Intro and history of deep learning
3	Deep learning optimization I
5	Convolutional deep learning
7	Graph deep learning
9	Multi-modal deep learning
11	What doesn't work in deep learning
13	Q&A

Lecture	Title
2	AutoDiff
4	Deep learning optimization II
6	Attention-based deep learning
8	From supervised to unsupervised deep learning
10	Generative deep learning
12	Non-Euclidean deep learning
14	Deep learning for videos

Learning and reflection

Understanding Deep Learning, Chapter 12

Thank you!