50.039 Theory and Practice of Deep Learning

W9-S3 Word Embedding Biases, Word Embedding Evaluation and Attention

Matthieu De Mari

About this week (Week 9)

1. Why are **embeddings** an essential component of Neural Networks (NNs)?
2. Why are **good embeddings** difficult to produce?
3. What are the **conventional approaches to embeddings in NLP?** What can we **learn from these approaches?**
4. What are the **typical issues with embeddings** and how do we address them?
5. **State-of-the-art** of current embedding problems, and **open questions** in research.

About this week (Week 9)

6. How do we evaluate the **quality/performance** of an embedding?
7. Can embeddings be **biased**?
8. Can we help the neural networks **identify the important parts of the context** to focus on?
9. What is **attention** in Neural Networks? What are **transformers** in Neural Networks?
10. What are the typical **uses for attention** these days?
11. What are the **limits of attention** and the **current research directions** on this topic?

In the last episode…

We have discussed the **embedding** problem, and its aspects in Natural Language Processing.

Many approaches can be used to obtain word embedding.

- **Unsupervised:** CBoW & SkipGram (2013), GloVe (2014), FastText (2018), ELMo (2018).
- **Supervised:** InferSent (2018).
- **Multi-task:** Google's Universal Word Embeddings (2019).

Question: What makes a word embedding better than another? How do we measure the performance of an embedding?

Measuring Embeddings Performance

- Evaluating the performance of an embedding is a difficult task and still an open question in research [Schnabel2015].

- **Reason:** it is very difficult to come up with a performance **measure** to objectively evaluate the quality of a word embedding.

Overall, **two approaches**:

- **Extrinsic methods**
- **Intrinsic Methods**

Measuring Embeddings Performance

Method #1 (Extrinsic Evaluation):

Methods of **extrinsic evaluation** are based on the ability of word embeddings to be used as the feature vectors of supervised machine learning algorithms.

The more able a word embedding is on a given (complex) task, the better it might be.

Measuring Embeddings Performance

Method #1 (Extrinsic Evaluation):

Methods of **extrinsic evaluation** are based on the ability of word embeddings to be used as the feature vectors of supervised machine learning algorithms.

The more able a word embedding is on a given (complex) task, the better it might be.

Typically, training a CBoW requires to predict a missing word, in the middle of a given sentence.

This is an easier task to train on, compared to predicting the possible context words for a given single word (SkipGram)

For this reason, we often assume that **SkipGram > CBoW.**

Measuring Embeddings Performance

Method #1 (Extrinsic Evaluation):

Methods of **extrinsic evaluation** are based on the ability of word embeddings to be used as the feature vectors of supervised machine learning algorithms.

The more able a word embedding is on a given (complex) task, the better it might be.

In general, we train an embedding on a task and extract a layer from our Neural Network, corresponding to the embedding.

This embedding is then reused on another task layer on (transfer learning).

For this reason, it is often preferable to check how good the embedding is on the second task.

Measuring Embeddings Performance

Method #1 (Extrinsic Evaluation):

Methods of **extrinsic evaluation** are based on the ability of word embeddings to be used as the feature vectors of supervised machine learning algorithms.

The more able a word embedding is on a given (complex) task, the better it might be.

Researchers have identified a set of typical NLP tasks and datasets on which we need word embeddings [Schnabel2015], e.g.

- Sentiment Analysis
- Translation
- Etc.

These have often been used as reference for measuring the performance of a given word embedding.

Measuring Embeddings Performance

Method #2 (Intrinsic Evaluation):

Methods of **intrinsic evaluation** are experiments in which word embeddings are compared with human judgments on words relations.

Typically, we will rely on **a human expert to judge** whether the word embedding is good or not.

Measuring Embeddings Performance

Definition (cosine angle between two vectors):

Researchers have tried to come up with an **"objective" metric** to compute the **similarity** between two word embeddings vectors.

For two vectors u, v, we define the **cosine angle** between both vectors as:

$$\cos(u,v) = \frac{\langle u,v \rangle}{\|u\|\|v\|}$$

The cosine angle formula verifies

$$\cos(u,v) = \begin{cases} 1 & if\ u = v \\ -1 & if\ u = -v \\ 0 & if\ u\ and\ v\ are\ orthogonal \end{cases}$$

A value close to 1 (resp. -1) indicate two vectors carrying a similar (resp. opposite) semantic meaning. A value 0 indicates two uncorrelated words.

Measuring Embeddings Performance

- **Rule #1: A "good" word embedding should model semantic proximity.**

One can check that word analogies are preserved by the word embedding, by e.g. looking for the top 10 words $(u_1, \dots, u_{10})$ with the largest cosine distance for any given word v.

These should make sense (somewhat)

Measuring Embeddings Performance

- **Rule #1: A "good" word embedding should model semantic proximity.**

One can check that word analogies are preserved by the word embedding, by e.g. looking for the top 10 words $(u_1, \ldots, u_{10})$ with the largest cosine distance for any given word v.

These should make sense (somewhat)

- For instance (from the FastText tutorial https://fasttext.cc/docs/en/unsupervised-tutorial.html)

```
Query word? asparagus
beetroot 0.812384
tomato 0.806688
horseradish 0.805928
spinach 0.801483
licorice 0.791697
lingonberries 0.781507
asparagales 0.780756
lingonberry 0.778534
celery 0.774529
beets 0.773984
```

```
Query word? pidgey
pidgeot 0.891801
pidgeotto 0.885109
pidge 0.884739
pidgeon 0.787351
pok 0.781068
pikachu 0.758688
charizard 0.749403
squirtle 0.742582
beedrill 0.741579
charmeleon 0.733625
```

Measuring Embeddings Performance

- **Rule #1: A "good" word embedding should model semantic proximity.**

One can check that word

Are these top10 lists good or bad answers?
Your answer to this question will probably be very subjective and not a reliable performance evaluation metric...!

- For instance (from the FastText tutorial https://fasttext.cc/docs/en/unsupervised-tutorial.html)

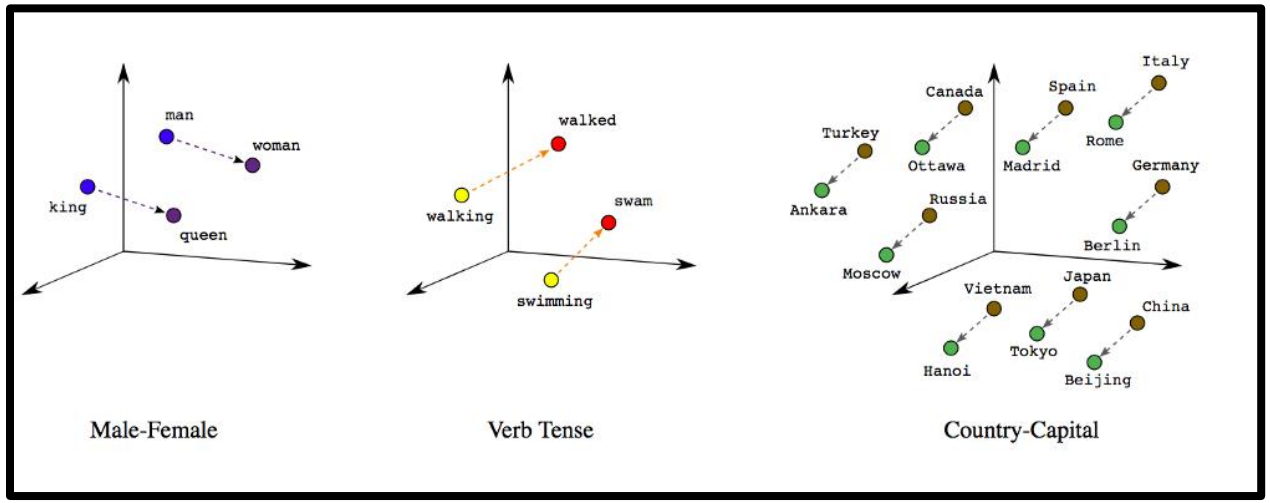
```
Query word? asparagus
beetroot 0.812384
tomato 0.806688
horseradish 0.805928
spinach 0.801483
licorice 0.791697
lingonberries 0.781507
asparagales 0.780756
lingonberry 0.778534
celery 0.774529
beets 0.773984
```

```
Query word? pidgey
pidgeot 0.891801
pidgeotto 0.885109
pidge 0.884739
pidgeon 0.787351
pok 0.781068
pikachu 0.758688
charizard 0.749403
squirtle 0.742582
beedrill 0.741579
charmeleon 0.733625
```

Measuring Embeddings Performance

- **Rule #2: A "good" word embedding should preserve word analogies.**

Word analogies are sentences reading as "A is to B, what C is to D", with A, B, C and D being four words.



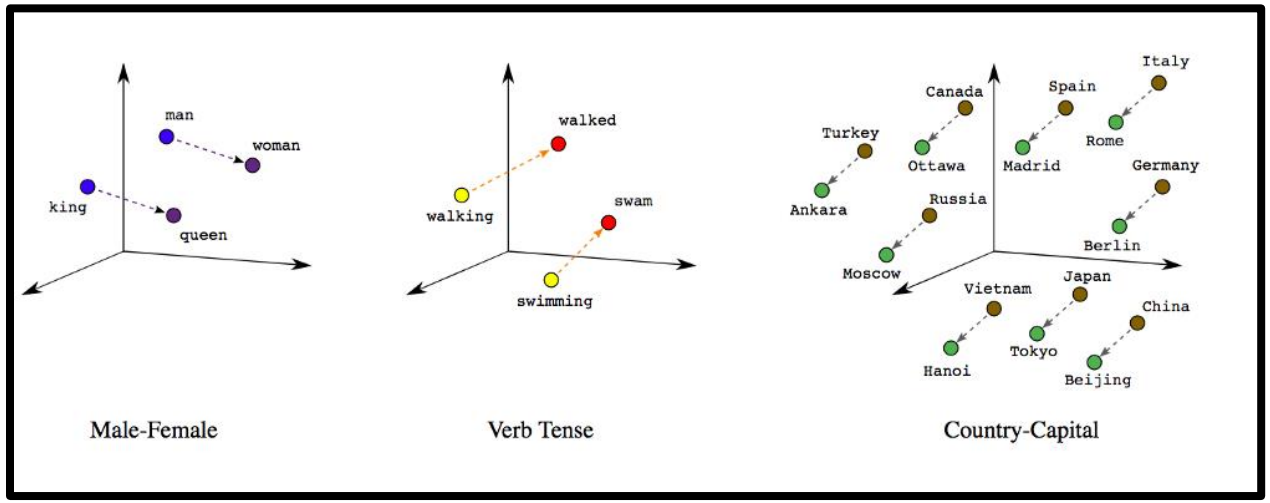
Measuring Embeddings Performance

- **Rule #2: A "good" word embedding should preserve word analogies.**

Word analogies are sentences reading as "A is to B, what C is to D", with A, B, C and D being four words.

Analogies are typically modeled by relations such as:

$$w_A - w_B = w_C - w_D$$
$$w_A = w_C - w_D + w_B$$



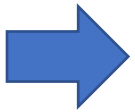
Measuring Embeddings Performance

- **Rule #2: A "good" word embedding should preserve word analogies.**

Word analogies are sentences

Is this top10 list containing good or bad answers?
Your answer to this question will (again!) probably be very subjective and not a reliable performance evaluation metric...!

- For instance (from the FastText tutorial https://fasttext.cc/docs/en/unsupervised-tutorial.html)



```
$ ./fasttext analogies result/fil9.bin
Pre-computing word vectors... done.
Query triplet (A - B + C)? berlin germany france
paris 0.896462
bourges 0.768954
louveciennes 0.765569
toulouse 0.761916
valenciennes 0.760251
montpellier 0.752747
strasbourg 0.744487
meudon 0.74143
bordeaux 0.740635
pigneaux 0.736122
```

Measuring Embeddings Performance

- **Rule #2: A "good" word embedding should preserve word analogies.**

Word analogies are sentences

Is this top10 list containing good or bad answers?
Your answer to this question will (again!) probably be very subjective and not a reliable performance evaluation metric...!

More specifically, because we are unsure **what the word embedding values represent in terms of semantics.**
There have been observed cases where the word embedding understands some "sort of relationship" but cannot properly distinguish between the different types of relationships, e.g.

$$\boldsymbol{w_{driver} - w_{car} = w_{king} - w_{queen}}$$

Measuring Embeddings Performance

- **Rule #3: A "good" word embedding should be able to operate on variations on the words (color/colour, typos, plural forms, conjugation, etc.).**

For instance (from the FastText tutorial https://fasttext.cc/docs/en/unsupervised-tutorial.html)

```
Query word? accomodation
accomodations 0.96342
accommodation 0.942124
accommodations 0.915427
accommodative 0.847751
accommodating 0.794353
accomodated 0.740381
amenities 0.729746
catering 0.725975
accomodate 0.703177
hospitality 0.701426
```

Measuring Embeddings Performance

- **Rule #3: A "good" word embedding should be able to operate on variations on the words (color/colour, typos, plural forms, conjugation, etc.).**

Is this top10 list containing good or bad answers?
Your answer to this question will (again!) probably be very subjective and not a reliable performance evaluation metric...!

For instance (from the FastText tutorial https://fasttext.cc/docs/en/unsupervised-tutorial.html)

```
Query word? accomodation
accomodations 0.96342
accommodation 0.942124
accommodations 0.915427
accommodative 0.847751
accommodating 0.794353
accomodated 0.740381
amenities 0.729746
catering 0.725975
accomodate 0.703177
hospitality 0.701426
```

Word embedding biases

- A quick note on word embeddings biases.
- As mentioned before, we value the quality of a word embedding by how good it manages to capture similarities and analogies between words, by extracting it from a given corpus.
- **Problem:** if the corpus is **biased** (sexist, racist, etc.), it will catch these **biases** as well [Bolukbasi2016].

Word embedding biases

$$w_{man\_job} - w_{man} + w_{woman} = ?$$

- A quick note on word embeddings biases.
- As mentioned before, we value the quality of a word embedding by how good it manages to capture similarities and analogies between words, by extracting it from a given corpus.
- **Problem:** if the corpus is **biased** (sexist, racist, etc.), it will catch these **biases** as well [Bolukbasi2016].

Extreme *she* occupations

1. homemaker	2. nurse	3. receptionist
4. librarian	5. socialite	6. hairdresser
7. nanny	8. bookkeeper	9. stylist
10. housekeeper	11. interior designer	12. guidance counselor

Extreme *he* occupations

1. maestro	2. skipper	3. protege
4. philosopher	5. captain	6. architect
7. financier	8. warrior	9. broadcaster
10. magician	11. figher pilot	12. boss

Word embedding biases

$$w_{man\_job} - w_{man} + w_{woman} = ?$$

- A quick note on word embeddings biases.
- As mentioned before, we value the quality of a word embedding by how good it manages to capture similarities and analogies between words, by extracting it from a given corpus.
- **Problem:** if the corpus is **biased** (sexist, racist, etc.), it will catch these **biases** as well [Bolukbasi2016].

Extreme *she* occupations

1. homemaker	2. nurse	3. receptionist
4. librarian	5. socialite	6. hairdresser
7. nanny	8. bookkeeper	9. stylist
10. housekeeper	11. interior designer	12. guidance counselor

Extreme *he* occupations

1. maestro	2. skipper	3. protege
4. philosopher	5. captain	6. architect
7. financier	8. warrior	9. broadcaster
10. magician	11. figher pilot	12. boss

Word embeddings evaluation

Question: What makes a word embedding better than another?

How do we measure the performance of an embedding?

How do we decide on the best word embedding to use?

Answer: Still an open question at the moment…

No perfect qualitative method for evaluating Word Embeddings.

More of a “consensus”, which changes over the years.

Still a lot of work needed to come up with a **“perfect” universal embedding** (if that is even possible?)

In the last episode…

We have discussed the **embedding** problem, and its aspects in Natural Language Processing.

Many approaches can be used to obtain word embedding.

- **Unsupervised:** CBoW & SkipGram (2013), GloVe (2014), FastText (2018), ELMo (2018).
- **Supervised:** InferSent (2018).
- **Multi-task:** Google's Universal Word Embeddings (2019).

All tasks we investigated were a sequence-to-singular output (CBoW) or singular input-to-sequence tasks (SkipGram).

Supervised and combined learning?

- Word2Vec, GloVe and ELMo approaches are commonly referred to as **unsupervised (or semi-supervised) approaches to embedding,** both relying on the **distributional hypothesis** we mentioned on the previous lecture.
- Unsupervised representation learning of sentences had been the norm for quite some time.
- **More advanced versions of unsupervised approaches exist!**

- However, the last few years have seen a shift toward **supervised** and **multi-task learning schemes** with a number of notably interesting proposals in late 2017/early 2018.

From unsupervised to combined learning

For a long time, there was a general consensus (see [Wang2018] for details) in the field that

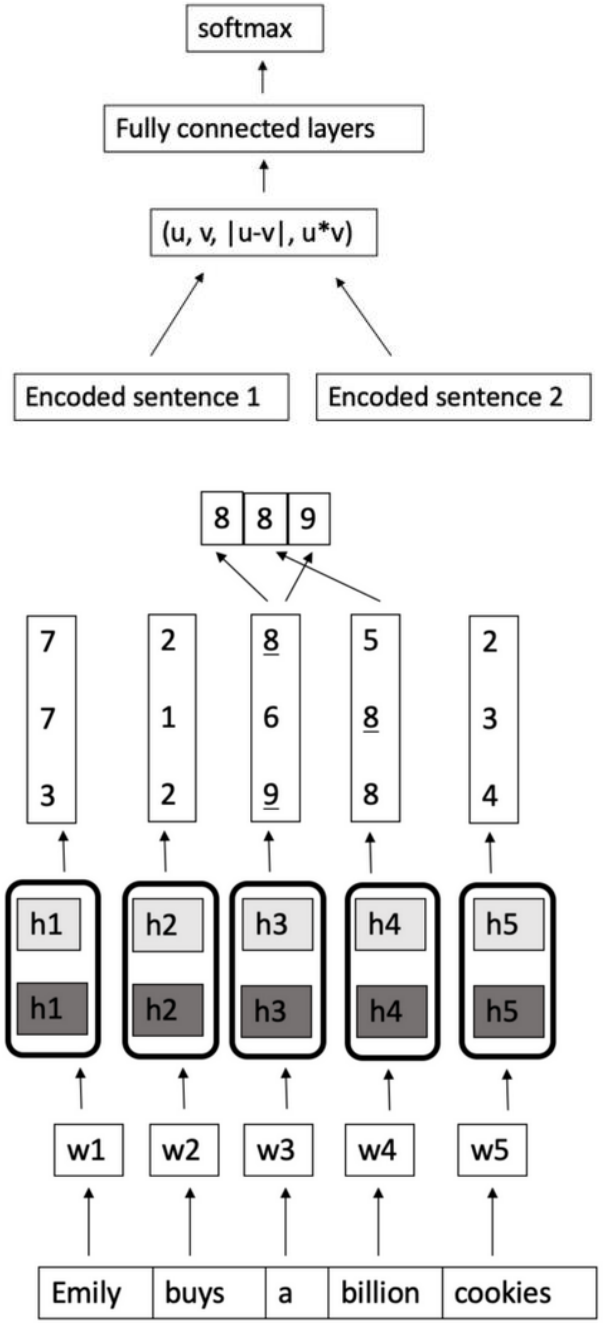
- the simple approach of **simply averaging a sentence's word vectors** (as in CBoW approach) gives a strong baseline for context, which is good enough for many downstream tasks.
- **supervised** learning of sentence/word embeddings was costly and gave lower-quality embeddings than unsupervised/semi-supervised approaches.

But these assumptions have recently been overturned, in part following the publication of the InferSent ([Conneau2018]) results.

Counter-example: InferSent (out-of-class)

Simple key takeaways

- The InferSent model (in [Conneau2018]) is an example of a supervised classification model, with three classes: Entailment, Neutral, Contradiction.
- It uses the same forward-backward LSTM architecture as ELMo, and attempts to compare two sentences and classify the pair.
- Feature representation extracted from model showed good performance.



From unsupervised to combined learning

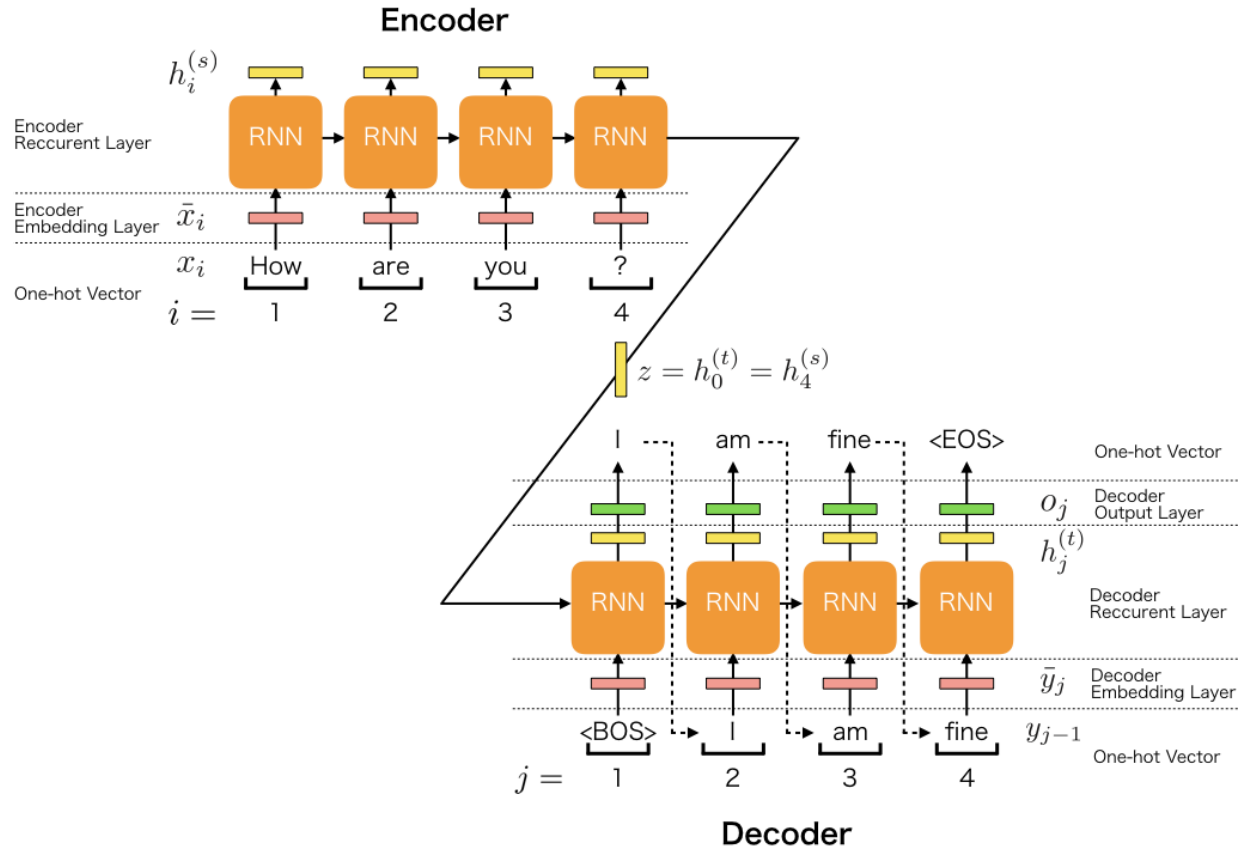
- Following these recent results, showing that supervised approaches are not necessarily bad…
- The current direction in research for word embeddings suggests to combine both supervised and unsupervised learning techniques. This is commonly referred to as a **multi-task learning based embedding.**
- The most notable work in this direction would be **Google's Universal Sentence and Word Encoder** (in [Cer2019]). Kept rather hidden, have only seen a pre-release on Tensorflow: https://www.tensorflow.org/hub/tutorials/semantic_similarity_with_tf_hub_universal_encoder
- (But nothing for PyTorch yet?)

Sequence to sequence tasks

Definition (Seq2Seq):

A **Sequence-to-Sequence task (Seq2Seq)** is a family of machine learning tasks which turns an input sequence into another output sequence.

For instance, language to language translation (French sentence into an English sentence), answers to questions Chatbots, etc.



Sequence to sequence tasks

Definition (Encoder-Decoder architecture):

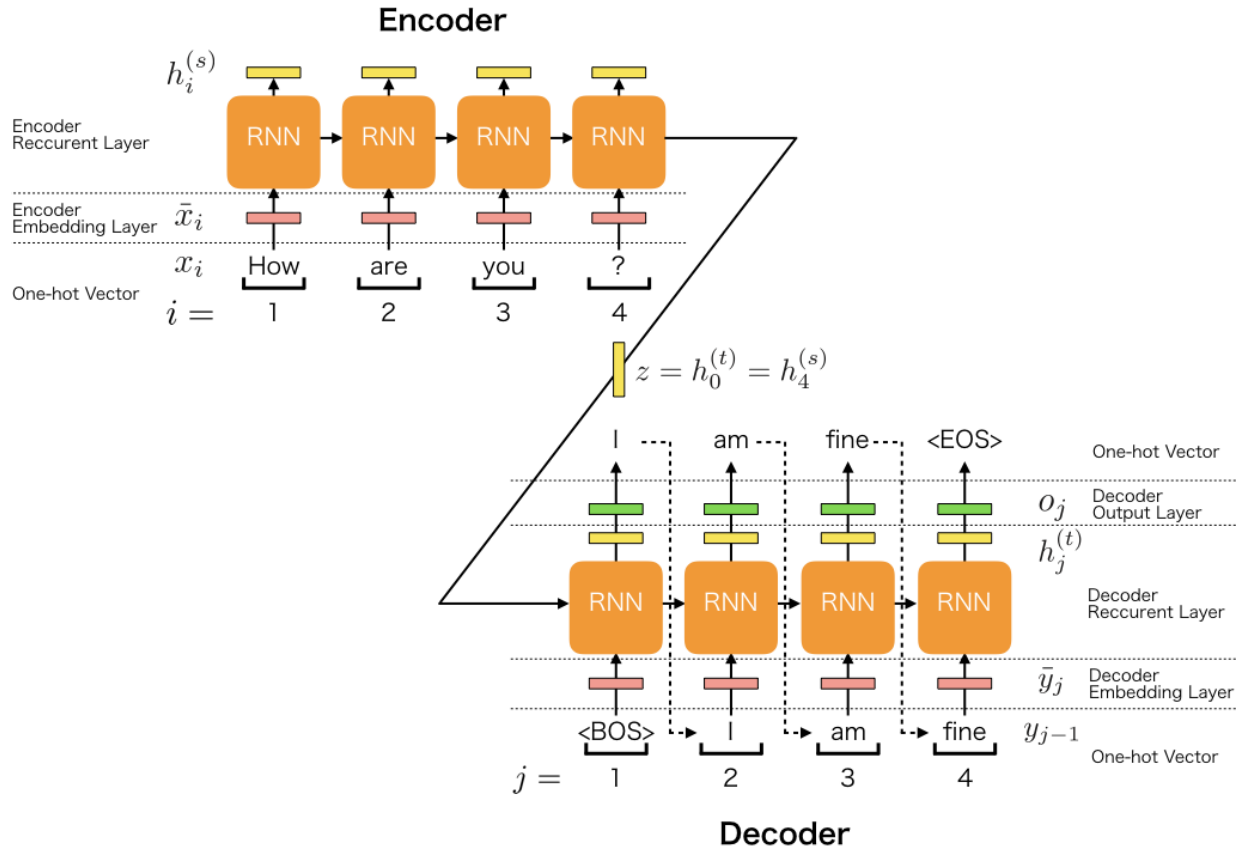
An **Encoder-Decoder architecture** typically consists of two parts:

- An **encoder**, which converts a sequence into an embedding vector, presumably encapsulating the **context** of the sequence.
- A **decoder**, which reads the embedding vector for **context** and constructs an output sequence.
- Both the encoder and the decoder will be modeled as RNNs as they both operate on sequences.

Sequence to sequence tasks

In the classical Seq2Seq architecture,

- a **word embedding** is used for each word of the sentence,
- An (often bi-directional) **encoder** RNN is used to propagate and encode context among words in the sentence,
- A **decoder** RNN reads the context and produces a new sentence.



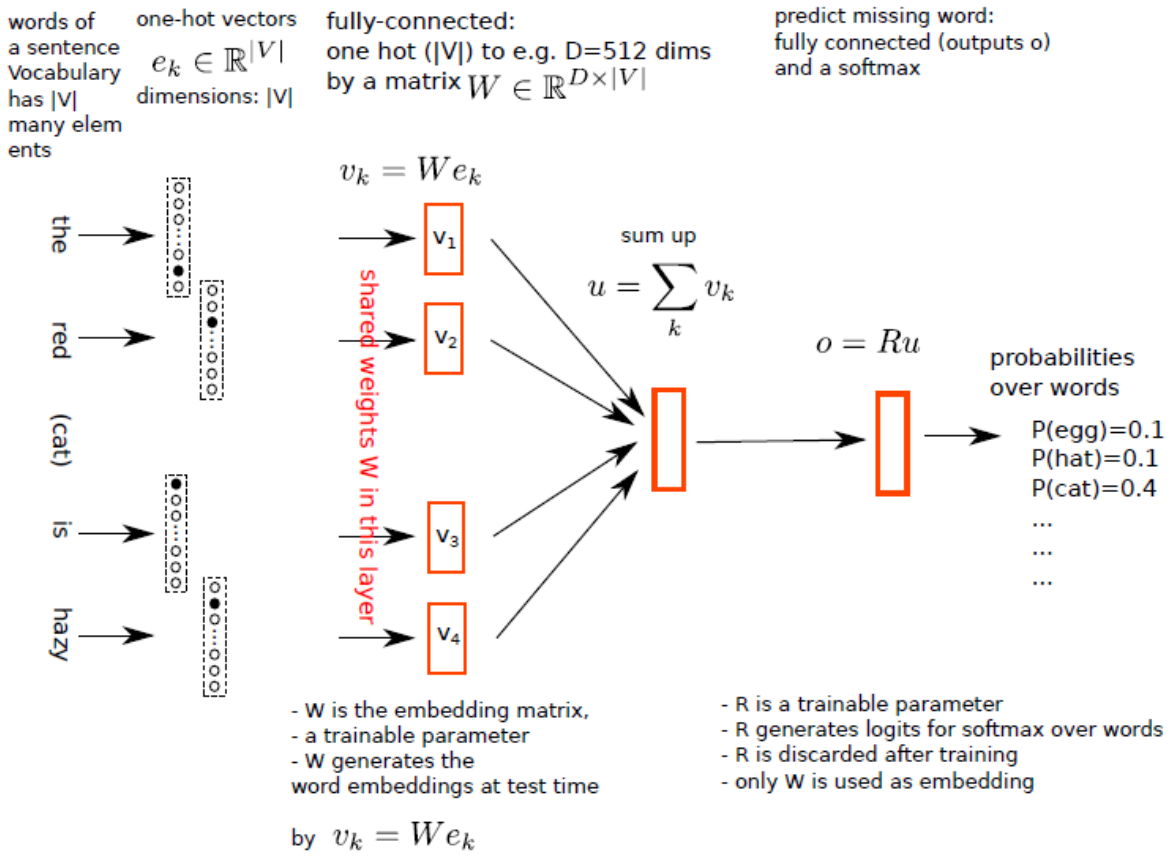
Sequence to sequence tasks

How is our context vector modeled again?

- **Averaging** the words embedding vectors for the words in the context (CBoW)

$$u = \sum_k v_k$$

Simple, but words might cancel each other out. Averaging effect if large number of words.



Sequence to sequence tasks

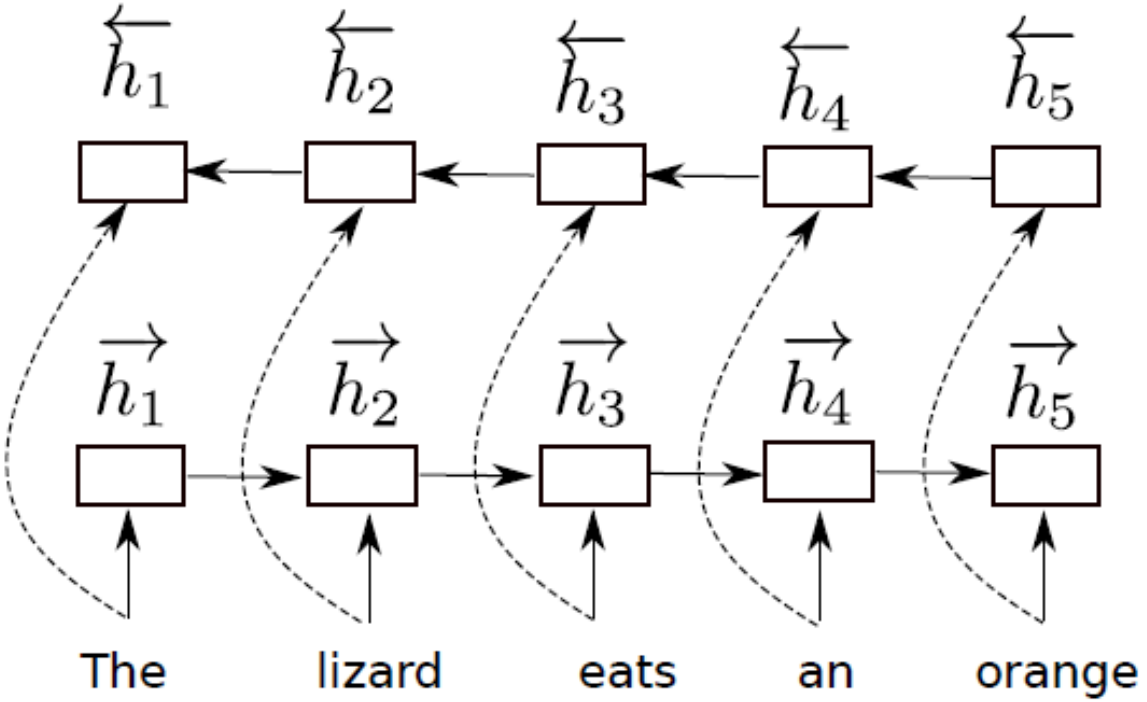
How is our context vector modeled again?

- **Concatenating** the hidden states of the bi-RNN gives surrounding context to word w_k (ELMo)

$$h_k = [\overleftarrow{h_k}, \overrightarrow{h_k}]$$

Preserves meaning of all surrounding words.

$$h_k = \left[\overleftarrow{h_k}, \overrightarrow{h_k}\right]$$



Sequence to sequence tasks

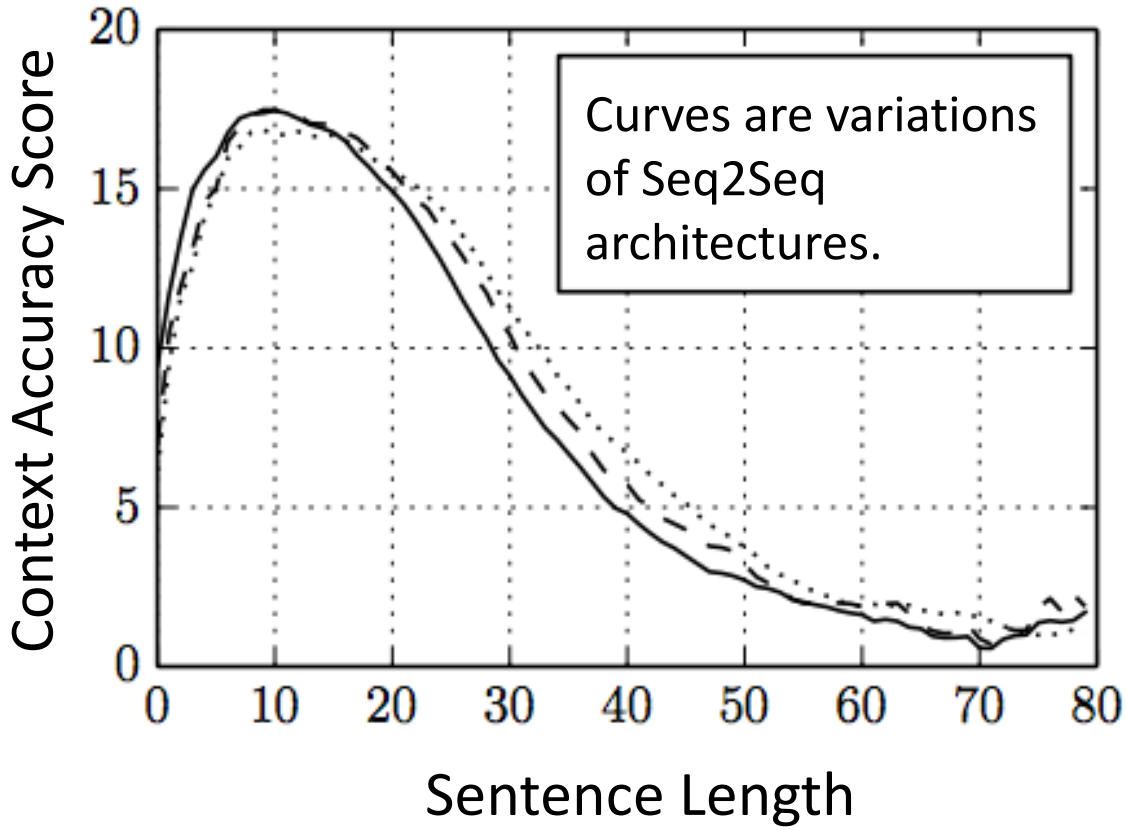
How is our context vector modeled again?

- **Concatenating** the words embedding vectors for the words in the context (ELMo)

$$h_k = [\overleftarrow{h_k}, \overrightarrow{h_k}]$$

Preserves meaning of all surrounding words.

Vanishing effect between the successive hidden states if large number of words [Vaswani2017].

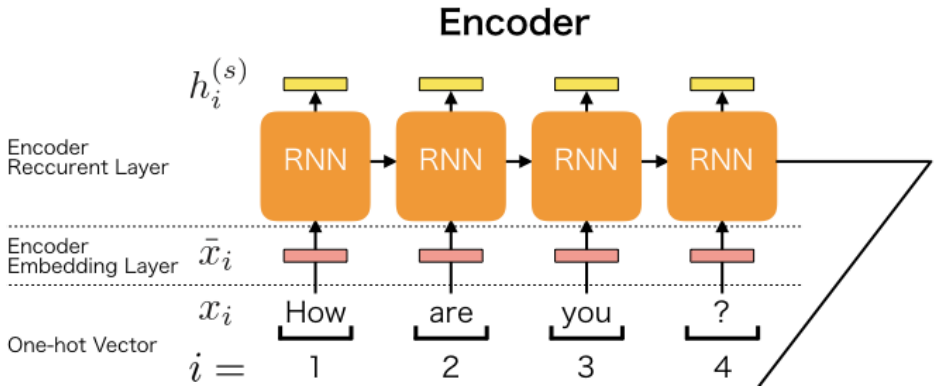


From: https://www.kaggle.com/sarthakvajpayee/attention-mechanism-japanese-english/notebook

Sequence to sequence tasks

What is the reason for the loss of context meaning in large sentences?

Words that are far away from each other will have their context lost by the hidden state of the LSTM RNN.



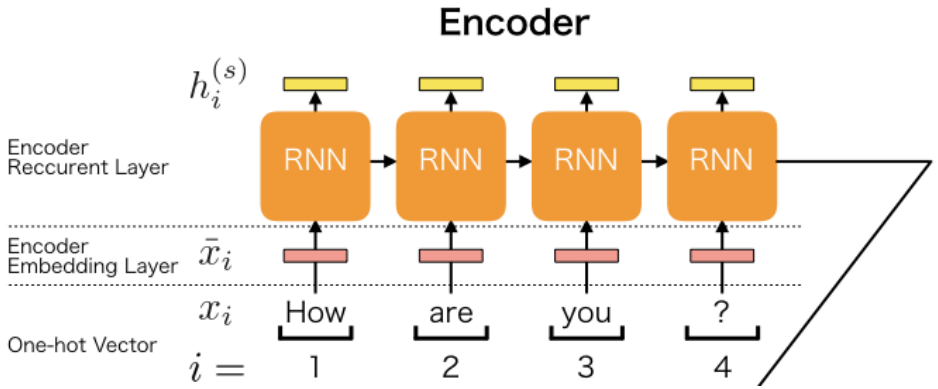
- For instance,

"Paris was a universe whole and entire unto herself, hollowed and fashioned by history; so she seemed in this age of Napoleon III with her towering buildings, her massive cathedrals, her grand boulevards, and ancient winding medieval streets–as vast and indestructible as nature itself. All was embraced by her, by her volatile and enchanted populace thronging the galleries, the theaters, the cafes, giving birth over and over to genius and sanctity, philosophy and war, frivolity and the finest art; so it seemed that if all the world outside her were to sink into darkness, what was fine, what was beautiful, what was essential might there still come to its finest flower."

Sequence to sequence tasks

What is the reason for the loss of context meaning in large sentences?

Words that are far away from each other will have their context lost by the hidden state of the LSTM RNN.



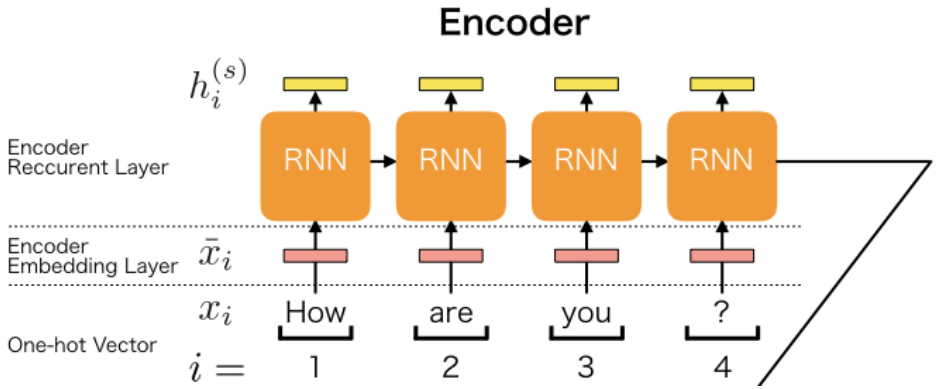
- For instance,

"Paris was a universe whole and entire unto herself, hollowed and fashioned by history; so she seemed in this age of Napoleon III with her towering buildings, her massive cathedrals, her grand boulevards, and ancient winding medieval streets–as vast and indestructible as nature itself. All was embraced by her, by her volatile and enchanted ***populace*** *thronging the galleries, the theaters, the cafes, giving birth over and over to genius and sanctity, philosophy and war, frivolity and the finest art; so it seemed that if all the world outside her were to sink into darkness, what was fine, what was beautiful, what was essential might there still come to its finest flower."*

Sequence to sequence tasks

What is the reason for the loss of context meaning in large sentences?

Words that are far away from each other will have their context lost by the hidden state of the LSTM RNN.



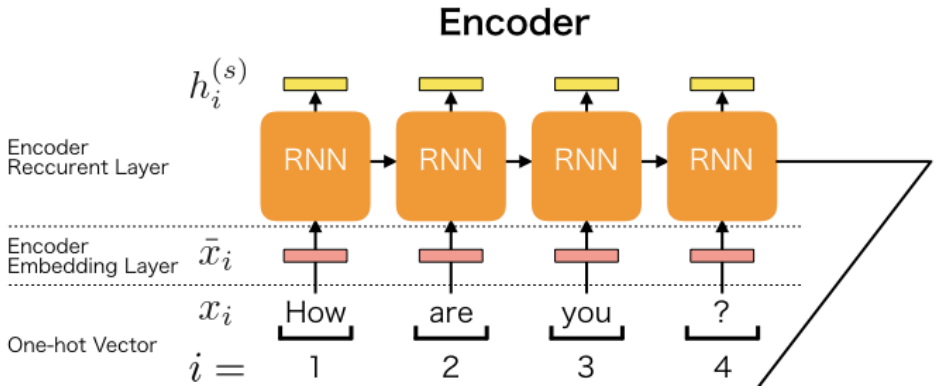
- For instance,

*"**Paris** was a universe whole and entire unto herself, hollowed and fashioned by history; so she seemed in this age of Napoleon III with her towering buildings, her massive cathedrals, her grand boulevards, and ancient winding medieval streets–as vast and indestructible as nature itself. All was embraced by her, by her volatile and enchanted **populace** thronging the galleries, the theaters, the cafes, giving birth over and over to genius and sanctity, philosophy and war, frivolity and the finest art; so it seemed that if all the world outside her were to sink into darkness, what was fine, what was beautiful, what was essential might there still come to its finest flower."*

Sequence to sequence tasks

What is the reason for the loss of context meaning in large sentences?

Words that are far away from each other will have their context lost by the hidden state of the LSTM RNN.



- Can we train our Neural Networks to get a better grasp of the important words that matter in the sequence and how to connect them, even when far away from each other?

- Yes! That is what the **attention layer** does!

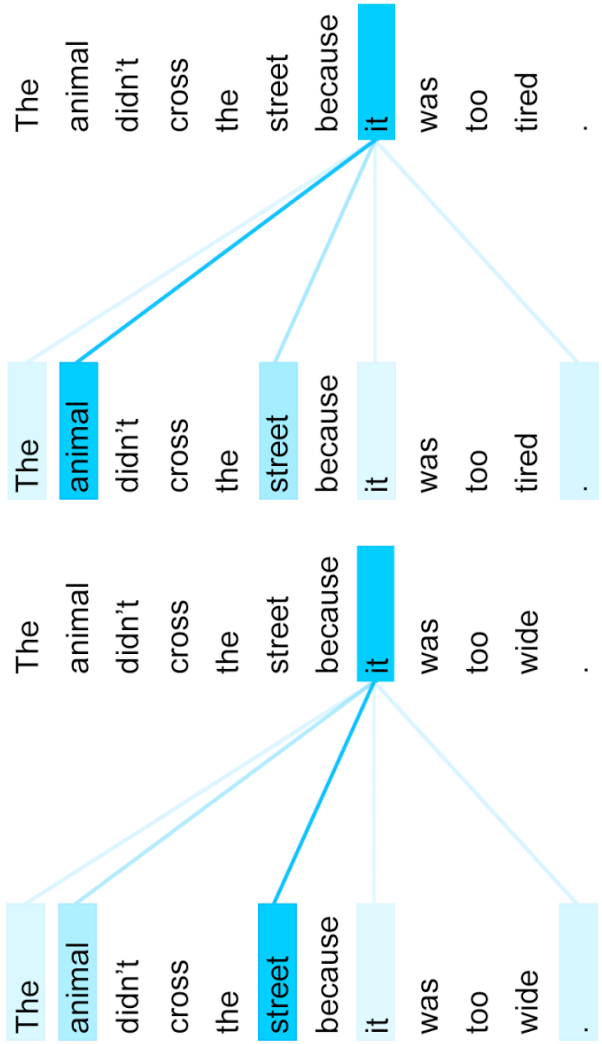
Attention: definition and motivation

Definition (attention):

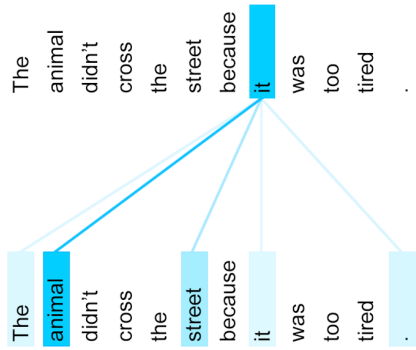
In the context of neural networks, **attention** (as introduced in [Vaswani2017]) is a technique that mimics cognitive attention.

The effect **enhances the important parts of the input data** and **fades out the rest**.

The thought being that **the network should devote more computing power on that small but important part** of the data.



Attention coefficients

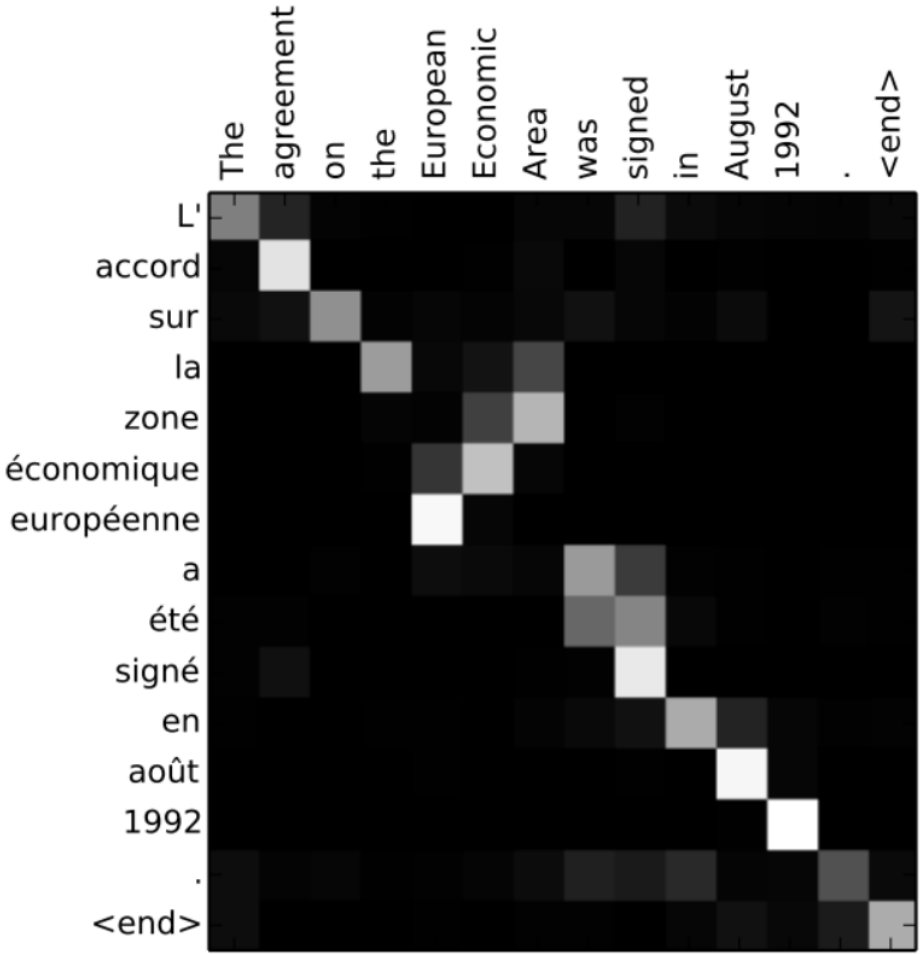


Definition (attention coefficient):

Attention coefficients identify which elements of the input sequence $x = (x_1, \dots, x_N)$ is relevant to compute an element y_k in the output sequence $y = (y_1, \dots, y_M)$.

$$a_{i,j} = importance\ of\ x_i\ to\ calculate\ y_j$$

For instance, attention helps with shenanigans such as adjective-noun placement in English and French.



Attention coefficients

Definition (attention coefficients):

Attention coefficients identify which elements of the input sequence $x = (x_1, \dots, x_N)$ is relevant to compute an element y_k in the output sequence $y = (y_1, \dots, y_M)$.

$$a_{i,j} = importance\ of\ x_i\ to\ calculate\ y_j$$

For instance, attention helps with shenanigans such as adjective-noun placement in English and French.

*"**Paris** was a universe whole and entire unto herself, hollowed and fashioned by history; so she seemed in this age of Napoleon III with her towering buildings, her massive cathedrals, her grand boulevards, and ancient winding medieval streets–as vast and indestructible as nature itself. All was embraced by her, by her volatile and enchanted **populace** thronging the galleries, the theaters, the cafes, giving birth over and over to genius and sanctity, philosophy and war, frivolity and the finest art; so it seemed that if all the world outside her were to sink into darkness, what was fine, what was beautiful, what was essential might there still come to its finest flower."*

Attention: definition and motivation

Definition (transformer):

A **transformer** network make extensive use of **attention mechanisms** and more specifically **attention layers** to achieve their expressive power.

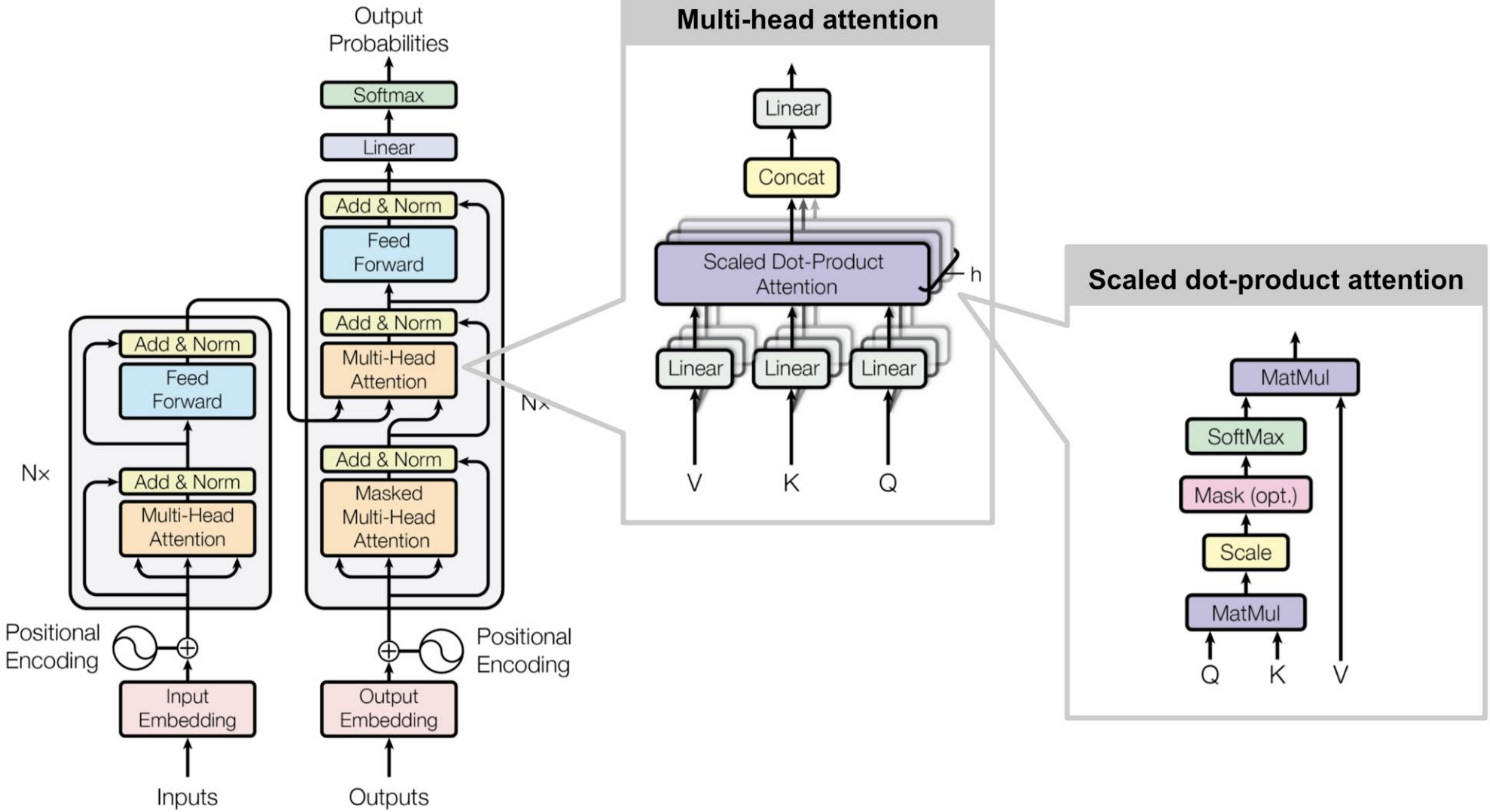
Transformers are currently considered as one of the most promising directions for Computer Vision, and Natural Language Processing.

Figure 27: Don’t you even think about it. Seriously, don’t.

The attention layer

Definition (the transformer architecture):

But first, a disclaimer from our beloved SUTD sponsor.


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

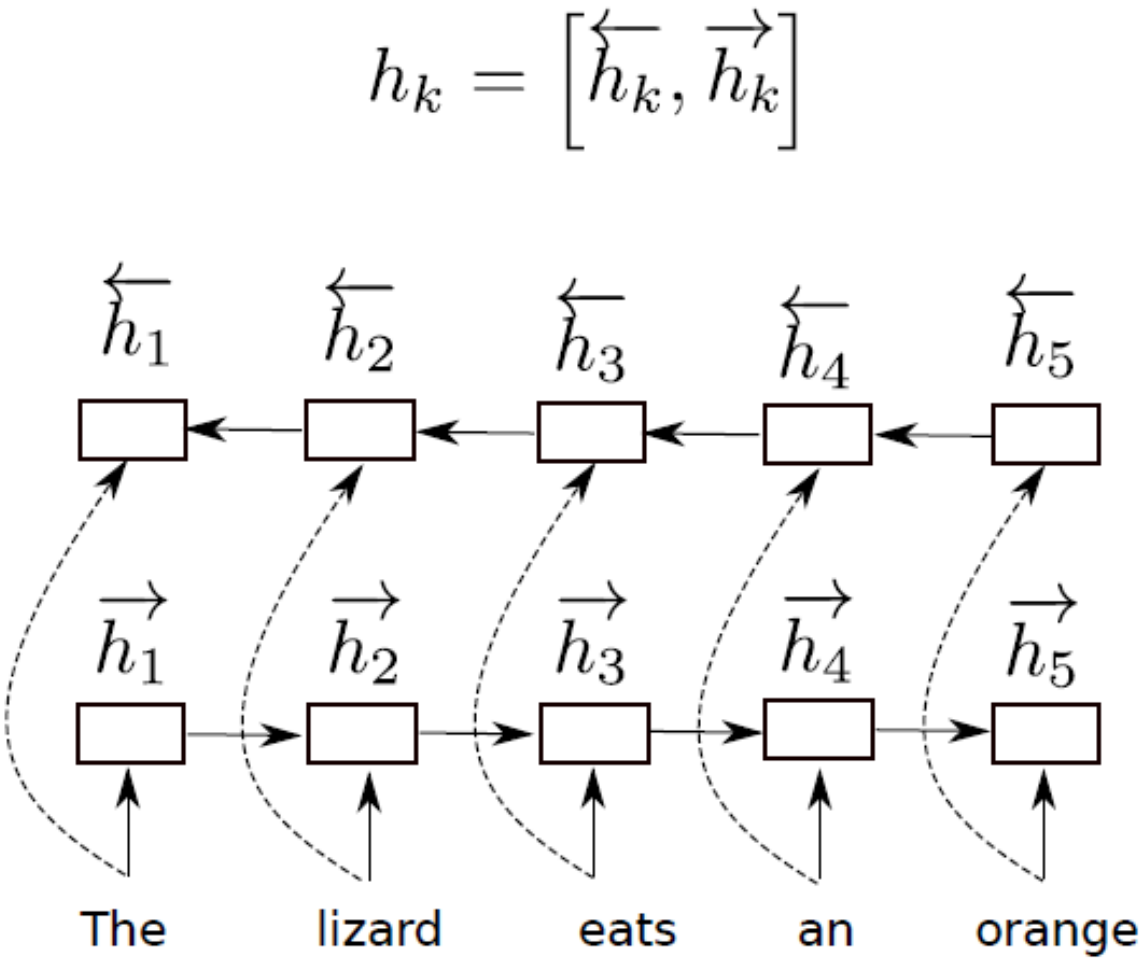
The attention layer

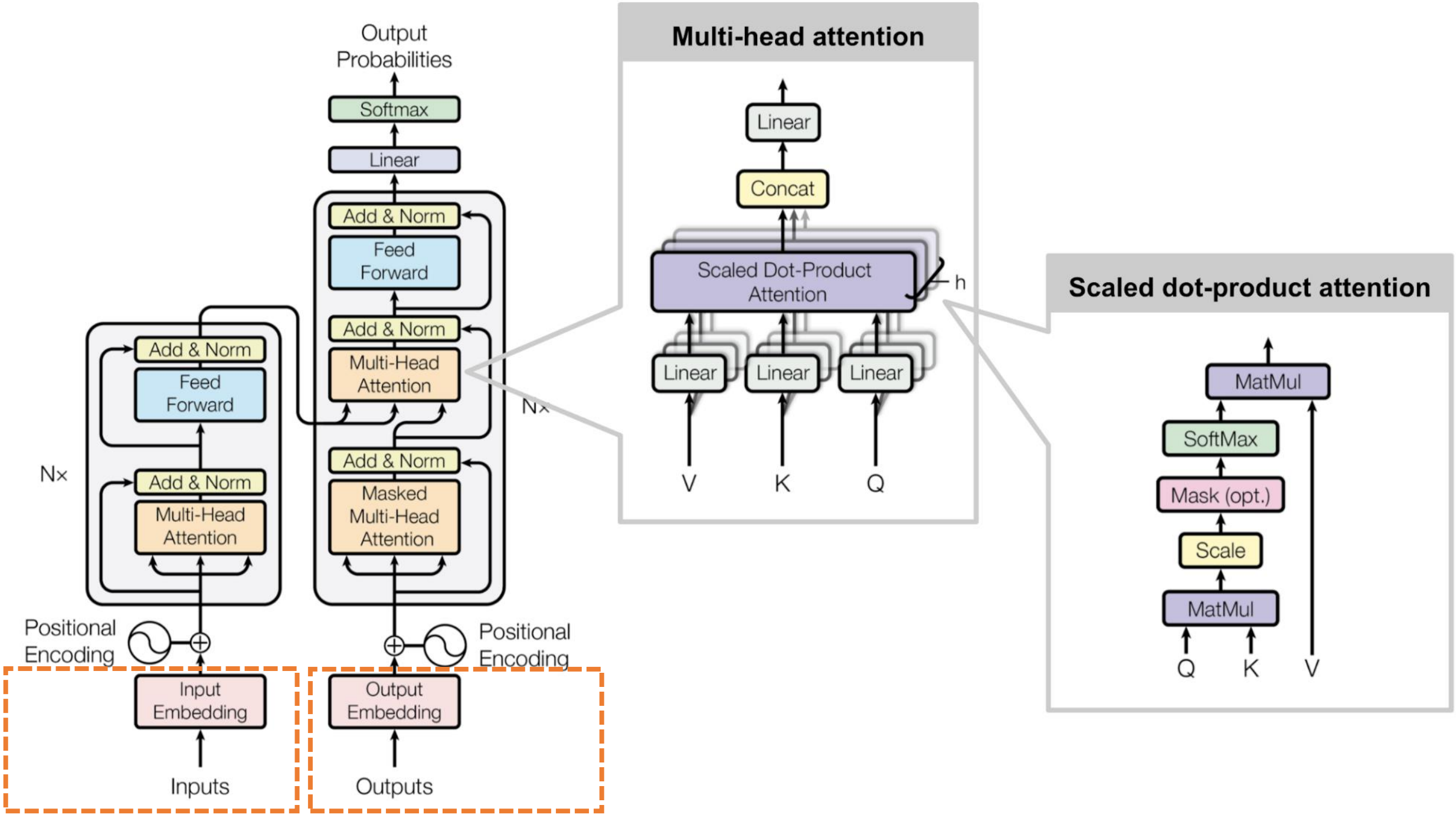
Definition (the transformer architecture):

1. It starts with a bidirectional RNN for input embedding, which is used to propagate context between words of an input sequence.

Obtain a first version of the context for any word w_k noted h_k (as in ELMo for instance).

$$h_k = [\overleftarrow{h_k}, \overrightarrow{h_k}]$$




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

The attention layer

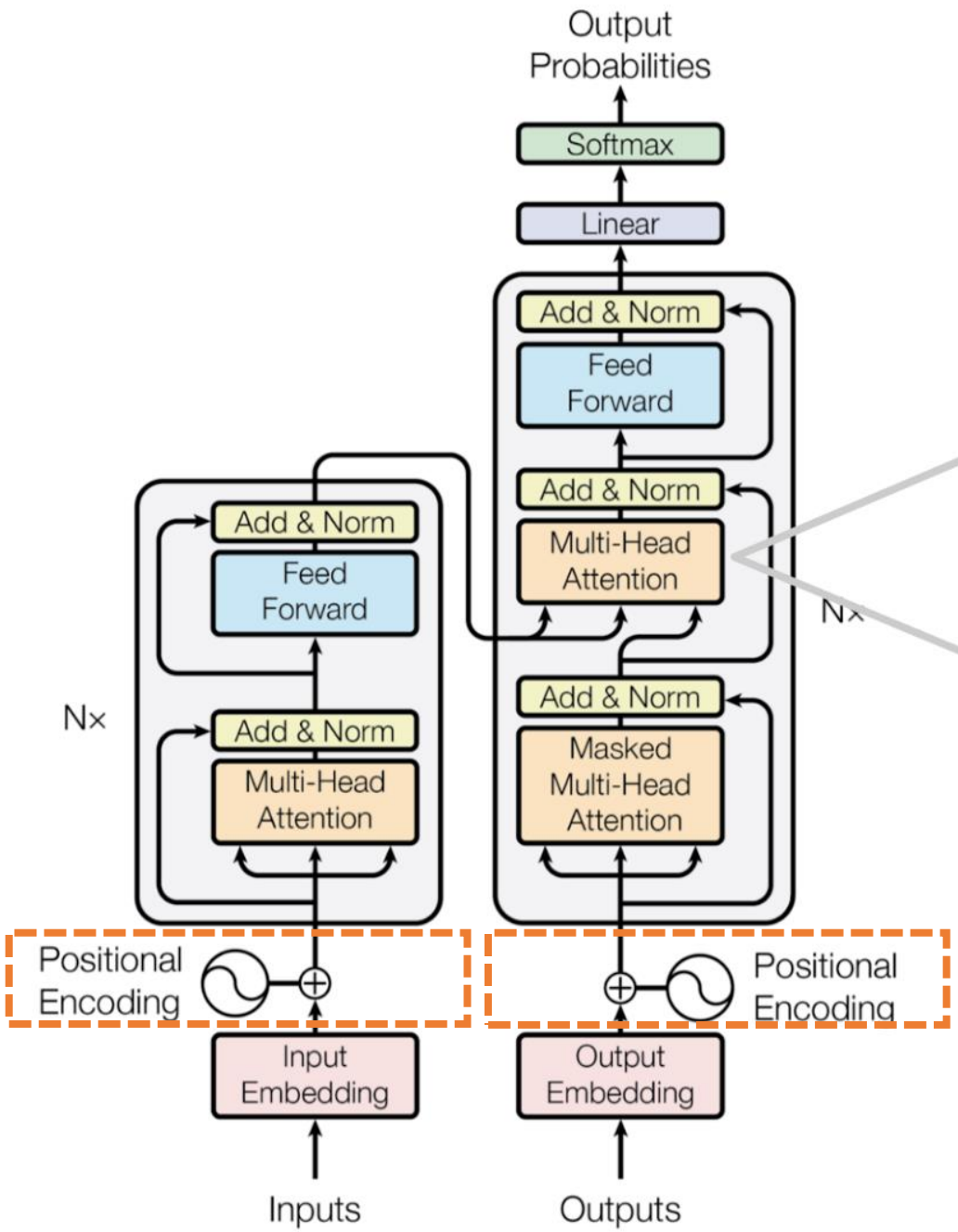
Definition (the transformer architecture):

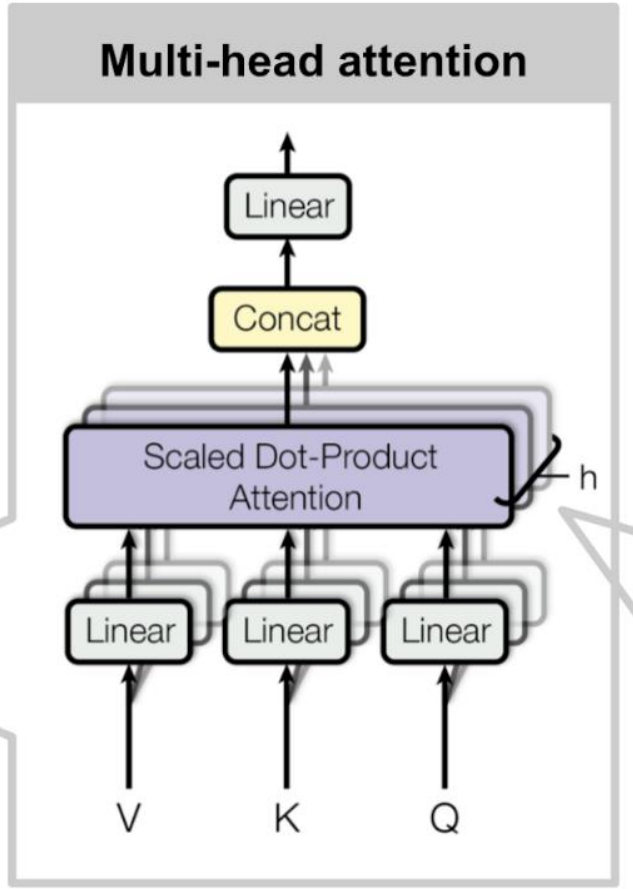
2. If you observe, the self-attention computations have no notion of ordering of words among the sequences.

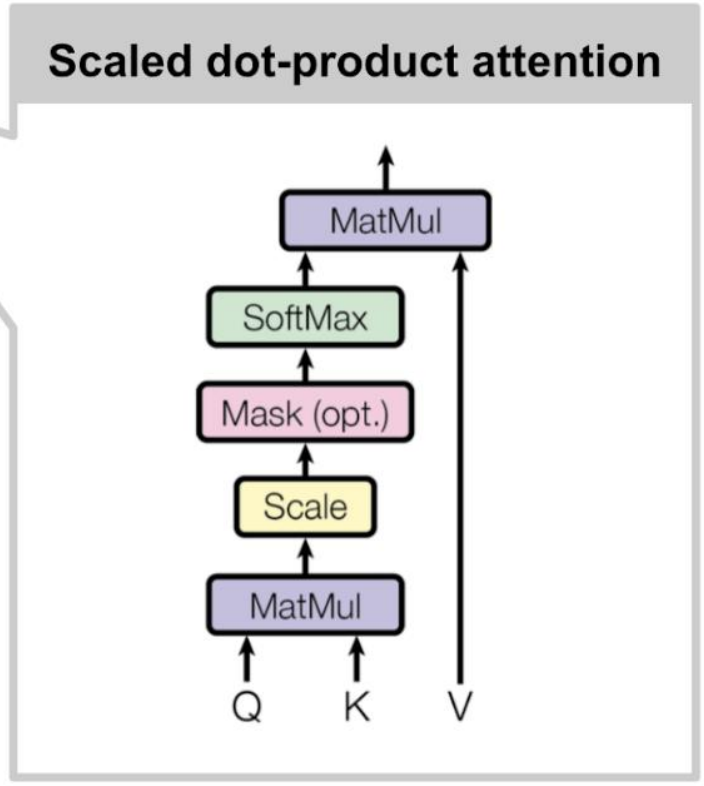
Although RNNs are slow, their sequential nature ensures that the order of words is preserved.

To elicit this notion of positioning of words in the sequence, Positional Encodings can be added to the regular input embeddings

(out of the scope of this class, will be skipped)


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs


Multi-head attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q


Scaled dot-product attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V

The attention layer

Definition (the multi-head self-attention layer a.k.a. the attention layer):

3. Producing vectors V, K, Q

Q is the query, i.e. the word I am currently looking at.

V and K refer to values and keys, and for simplicity will first consist of all the words you have in your sentence.

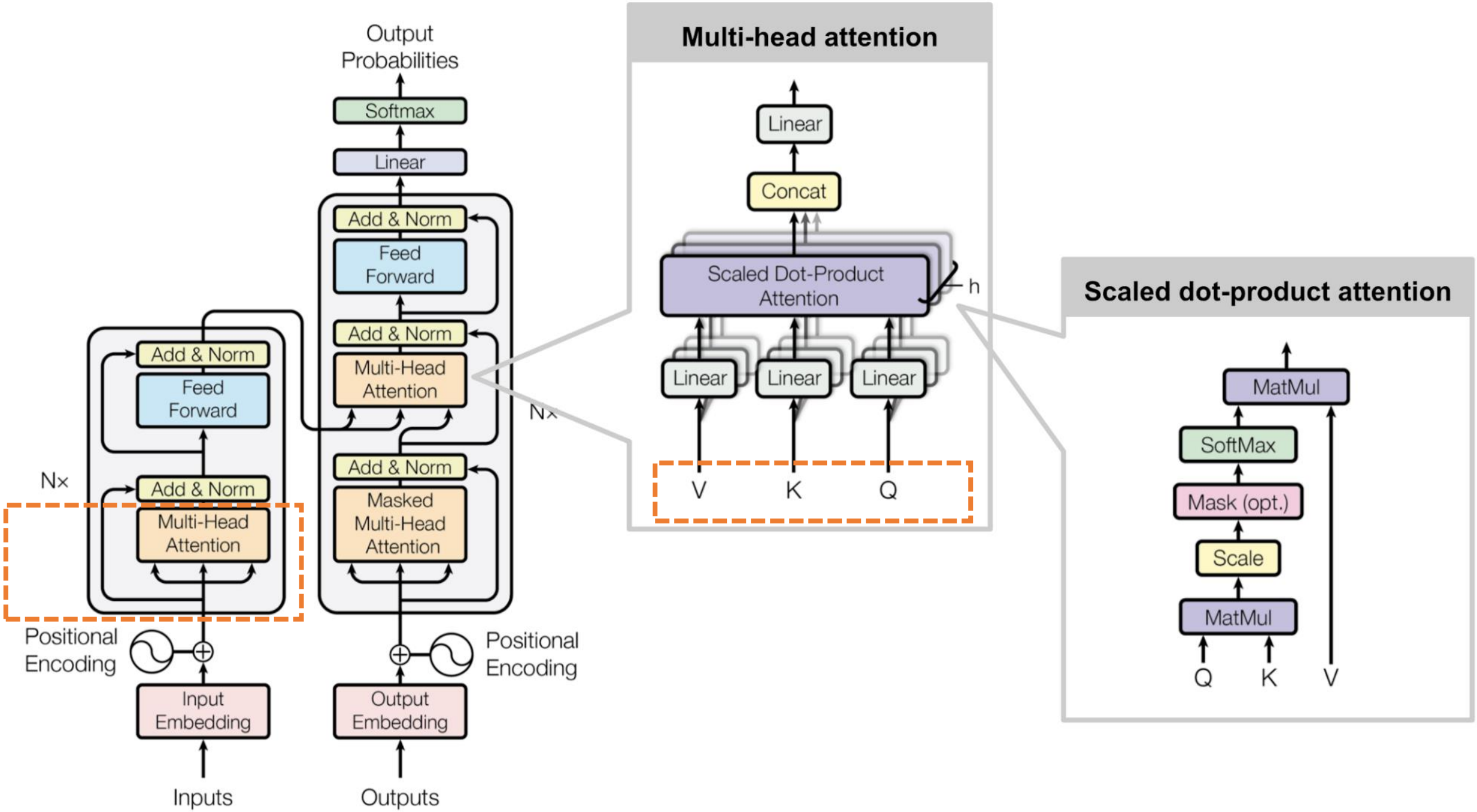
Let us consider the sentence

"Paris is the capital city of France."

When I multiply

- the context vector of the word city (i.e. one query word q_i),
- with another word of the sentence, e.g. Paris (i.e. one key word k_j),
- we obtain a similarity score, as in the dot-product/cosine angle formula from earlier.
- Repeat for all query/key words.

$$S = QK^T$$


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

The attention layer

Definition (the multi-head self-attention layer a.k.a. the attention layer):

3. Producing vectors V, K, Q

Q is the query, i.e. the word I am currently looking at.

V and K refer to values and keys, and for simplicity will first consist of all the words you have in your sentence.

Applying a softmax gives you a distribution over these similarities

$$S = QK^T$$
$$S_2 = softmax(S)$$

In general, we also like to normalize the QK^T multiplication, as in the cosine angle.

$$S = \frac{QK^T}{d}$$
$$S_2 = softmax(S)$$

The attention layer

Definition (the multi-head self-attention layer a.k.a. the attention layer):

3. Producing vectors V, K, Q

Q is the query, i.e. the word I am currently looking at.

V and K refer to values and keys, and for simplicity will first consist of all the words you have in your sentence.

Multiply this vector of distribution with the words in V, and you obtain a matrix of attention A!

$$A = softmax\left(\frac{QK^T}{d}\right)V$$

This matrix A encapsulates the inter dependencies between the words of your sentence!

The attention layer

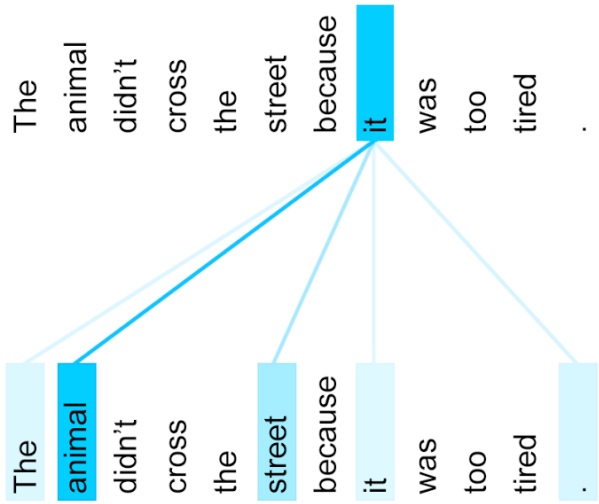
Definition (the multi-head self-attention layer a.k.a. the attention layer):

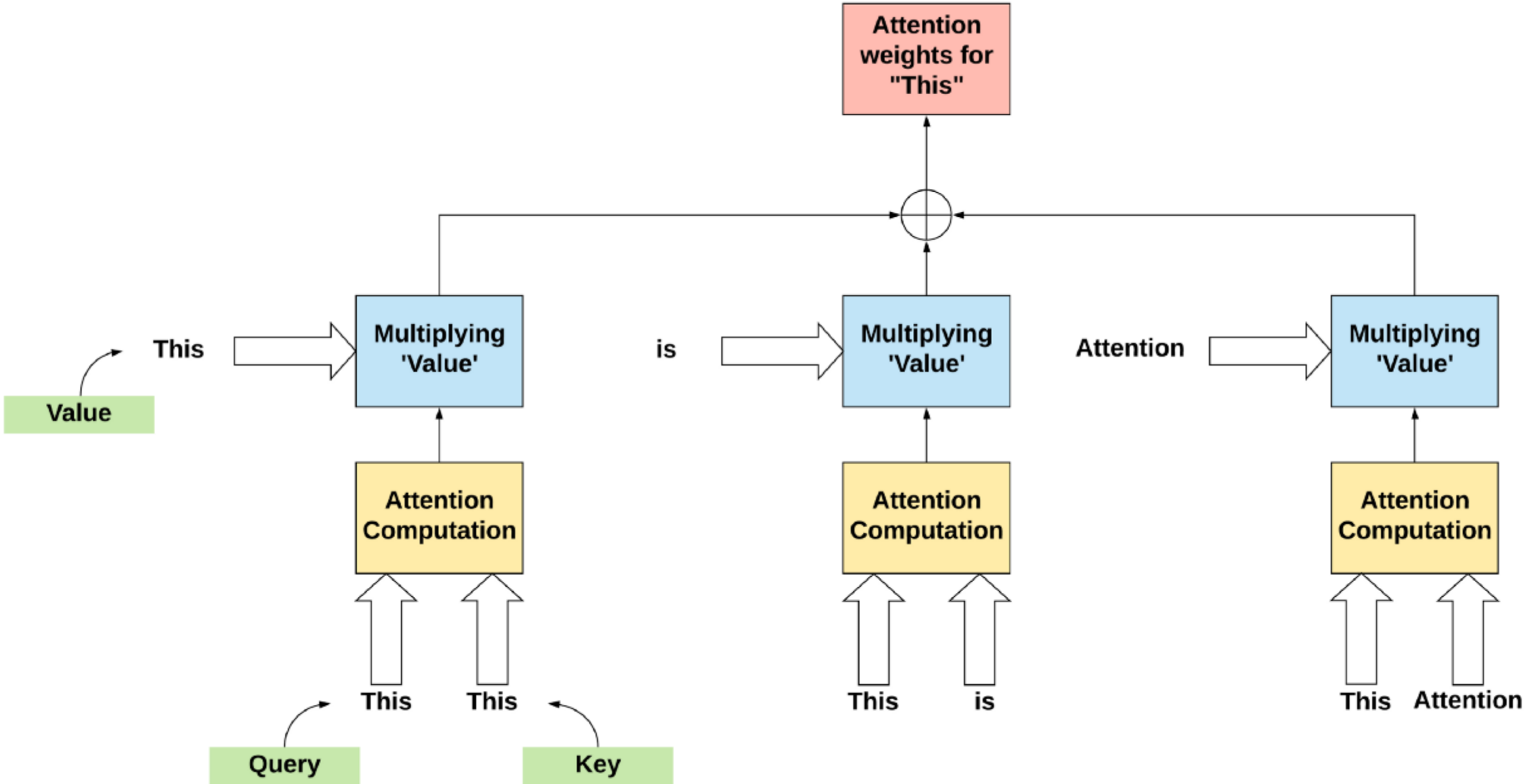
3. Producing vectors V, K, Q

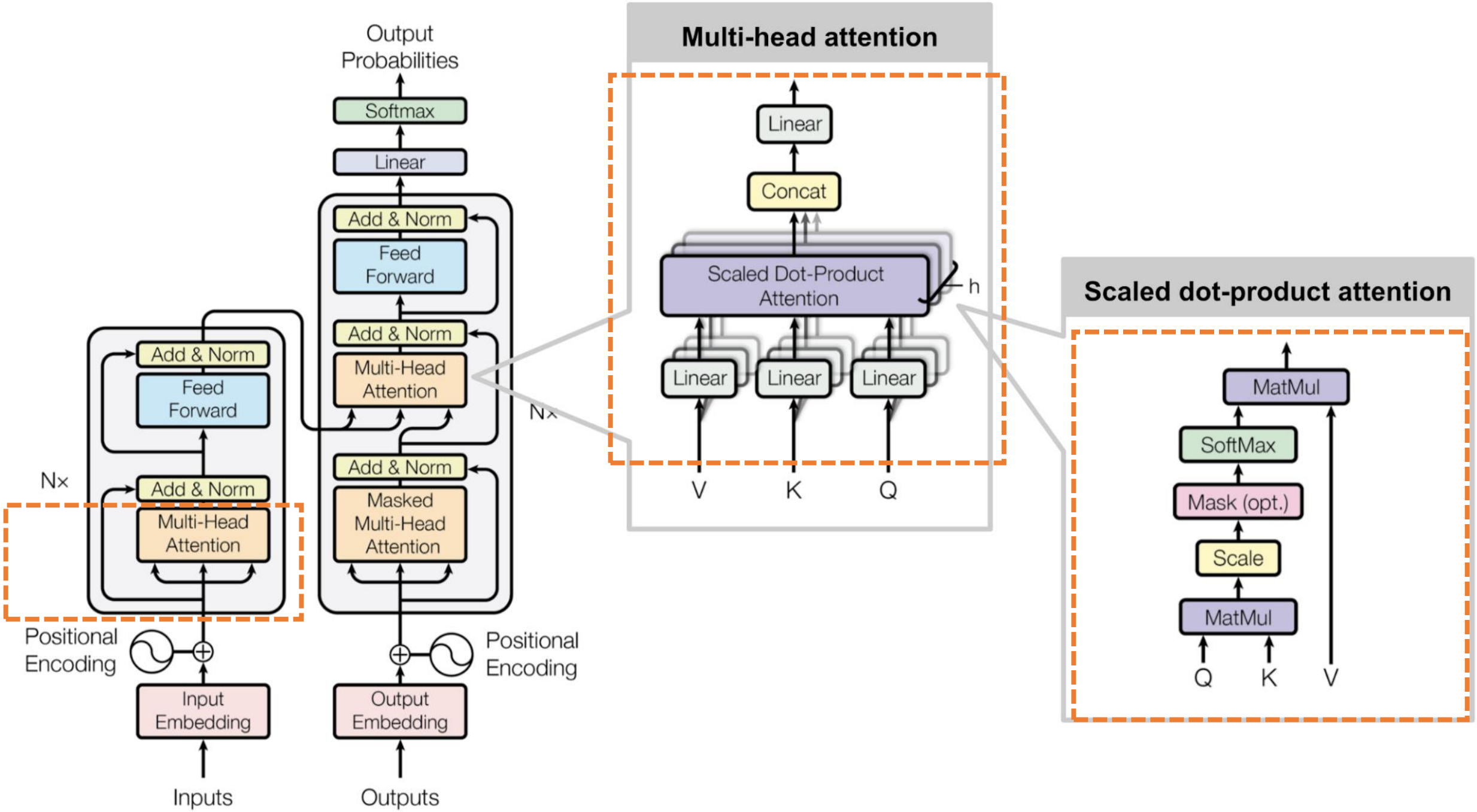
Q is the query, i.e. the word I am currently looking at.

V and K refer to values and keys, and for simplicity will first consist of all the words you have in your sentence.

This matrix S is often referred to as **self-attention**, as it gives inter-dependencies between the words of a single sentence.



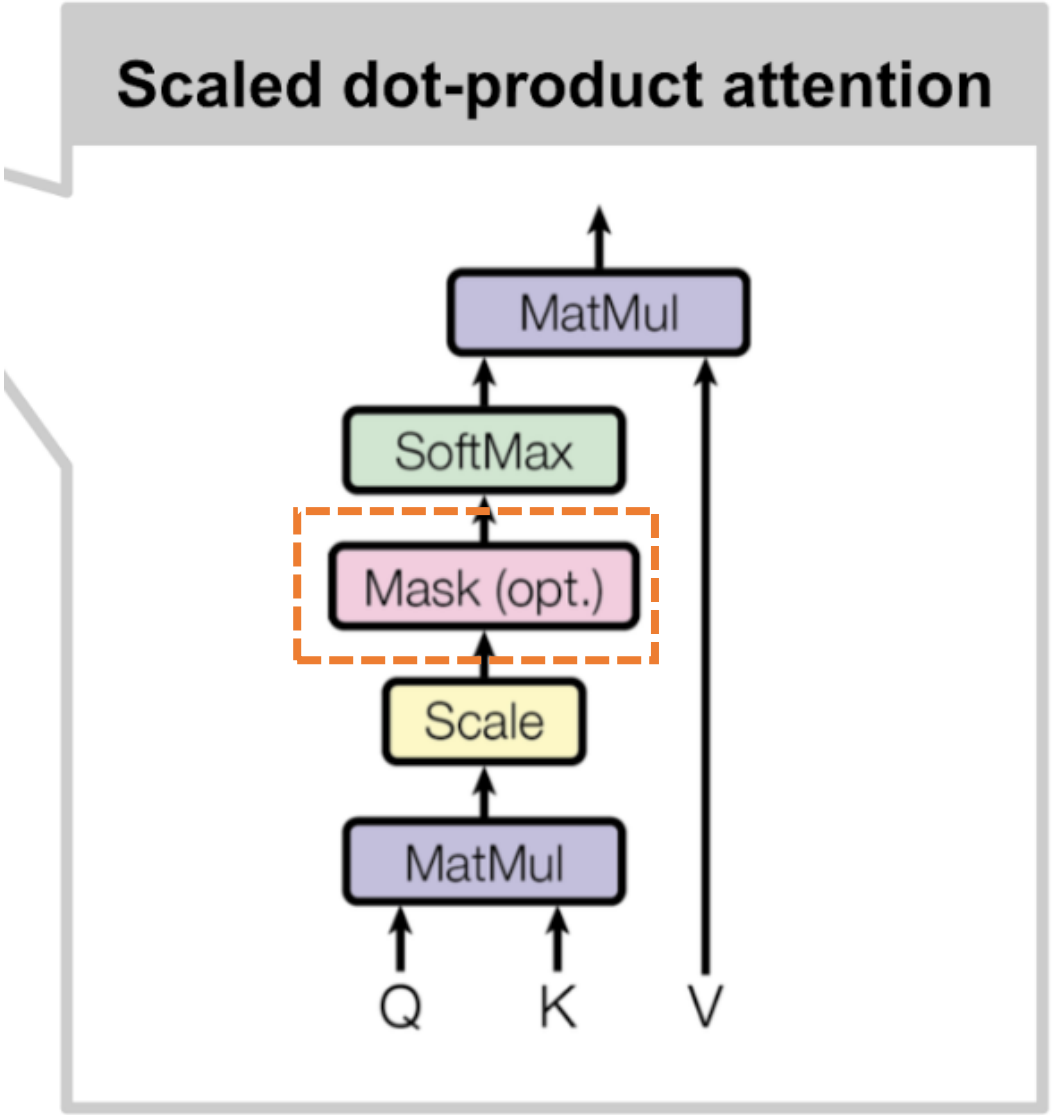


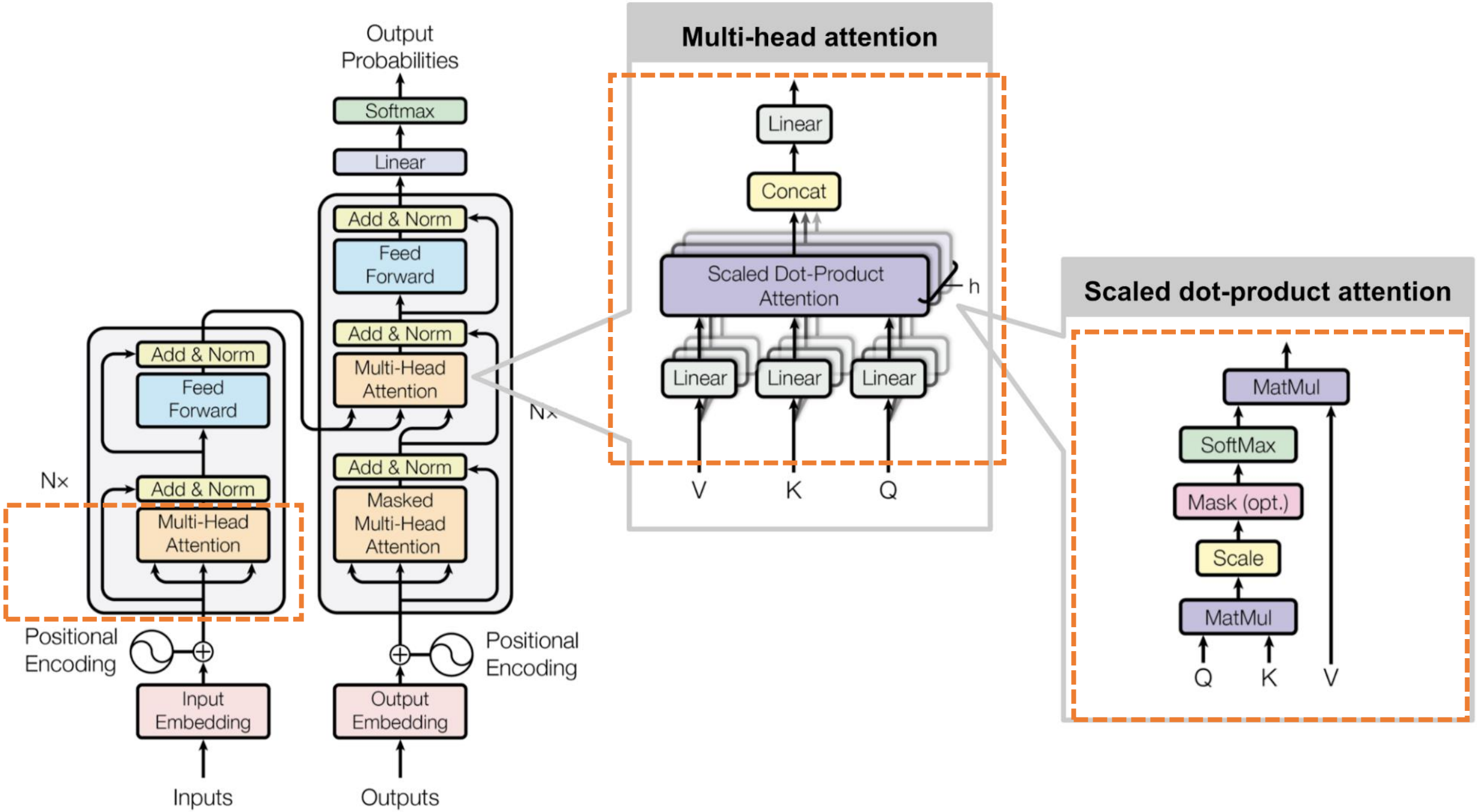

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs
Multi-head attention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

The attention layer

Definition (the multi-head self-attention layer a.k.a. the attention layer):

4. Masking is only used in the decoder part (and it will be discussed later).




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

```python
class SelfAttention(nn.Module):
    """
    Description: SelfAttention layer Class, describing the attention layers to
    be used in the transformer.
    Attributes list:
    - d_model: An integer, defining the dimension of the attention layer
    - output_size: An integer, defining the dimension of the output
    for the attention layer.
    - droupout_rate: A float value between 0 and 1, corresponding to the Dropout rate
    used in the Dropout layers of the Feed Forward layers.
    - mask: A boolean. If set to True, a triangular mask will be applied.
    """

    def __init__(self, d_model, output_size, dropout_rate = 0.3, mask = None):
        """
        Init Method for attention layer, mostly defining attributes.
        """
        super().__init__()
        self.query = nn.Linear(d_model, output_size)
        self.key = nn.Linear(d_model, output_size)
        self.value = nn.Linear(d_model, output_size)
        self.dropout = nn.Dropout(dropout_rate)
        self.mask = mask
```

```
    def forward(self, q, k, v):
        """
        Forward pass for the attention layer.
        """

        # Compute query, key and value parameters
        q_shape = q.shape[0]
        y_len = q.shape[1]
        seq_len = k.shape[1]
        query_out = self.query(q)
        key_out = self.key(k)
        value_out = self.value(v)
        key_dim = key.size(-1)
        transposed_key_out = key.transpose(1,2)

        # Calculate attention scores
        scores = torch.bmm(query, transposed_key_out)/np.sqrt(key_dim)
```

(mask part to be discussed later)

```
        # Final touches
        weights = F.softmax(scores, dim = -1)
        out = torch.bmm(weights, value_out)
        return out
```

```python
class MultiHeadAttentionLayer(nn.Module):
    """
    Description: MultiHeadAttention layer Class, describing the attention layers to
    be used in the transformer.
    Attributes list:
    - d_model: An integer, defining the dimension of the attention layer
    - num_heads: An integer, defining the number of attention heads
    to be used in both the encoder and decoder layers.
    - droupout_rate: A float value between 0 and 1, corresponding to the Dropout rate
    used in the Dropout layers of the Feed Forward layers.
    - mask: A boolean. If set to True, a triangular mask will be applied.
    """

    def __init__(self, d_model, num_heads, dropout_rate, mask = None):
        """
        Init method, mostly defining attributes.
        """
        super().__init__()
        self.d_model = d_model
        self.num_heads = num_heads
        self.droupout_rate = droupout_rate
        self.attention_output_size = d_model/num_heads
        layers = [SelfAttention(d_model, self.attention_output_size, dropout_rate, mask) \
                  for i in range(num_heads)]
        self.attention_layers = nn.Module(layers, )
        self.final_layer = nn.Linear(d_model, d_model)
```

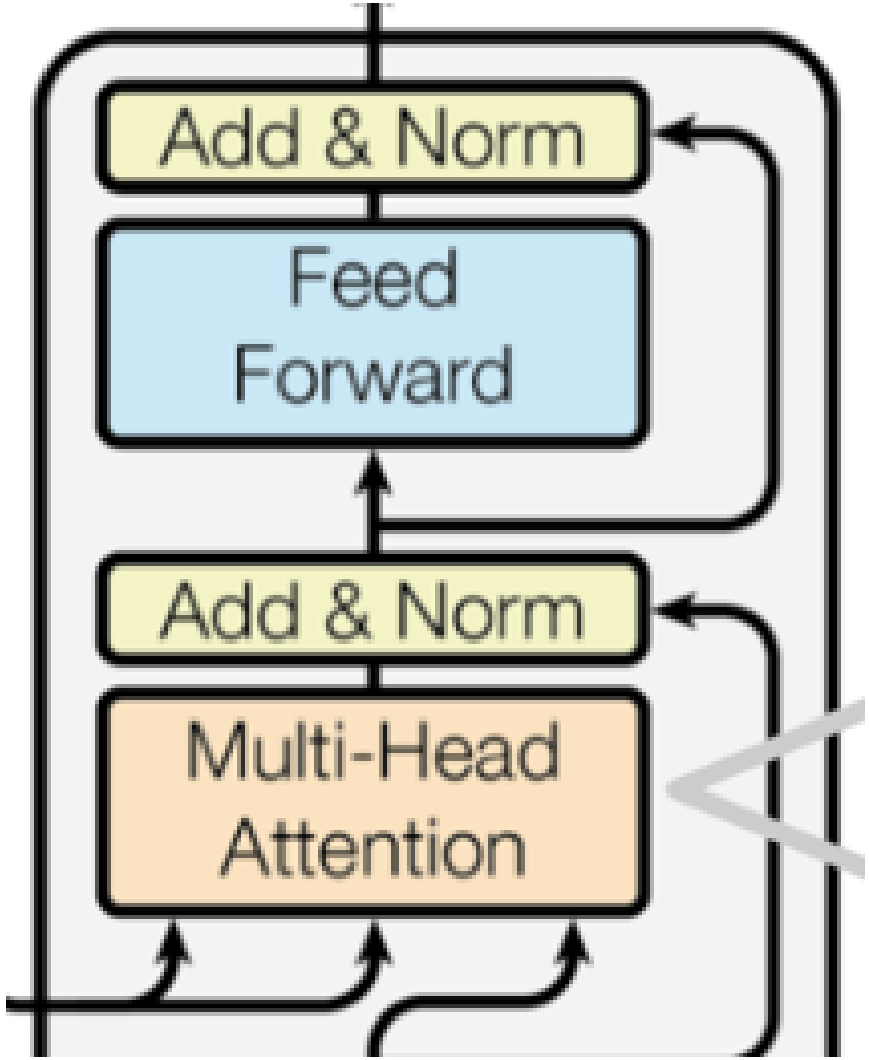
```
    def forward(self, q, k, v):
        """
        Forward pass, simply going through each attention layer.
        Applying one linear for final touch.
        """
        x = torch.cat([layer(q, k, v) for layer in self.attention_layers], dim = -1)
        x = self.final_layer(x)
        return x
```

The attention layer

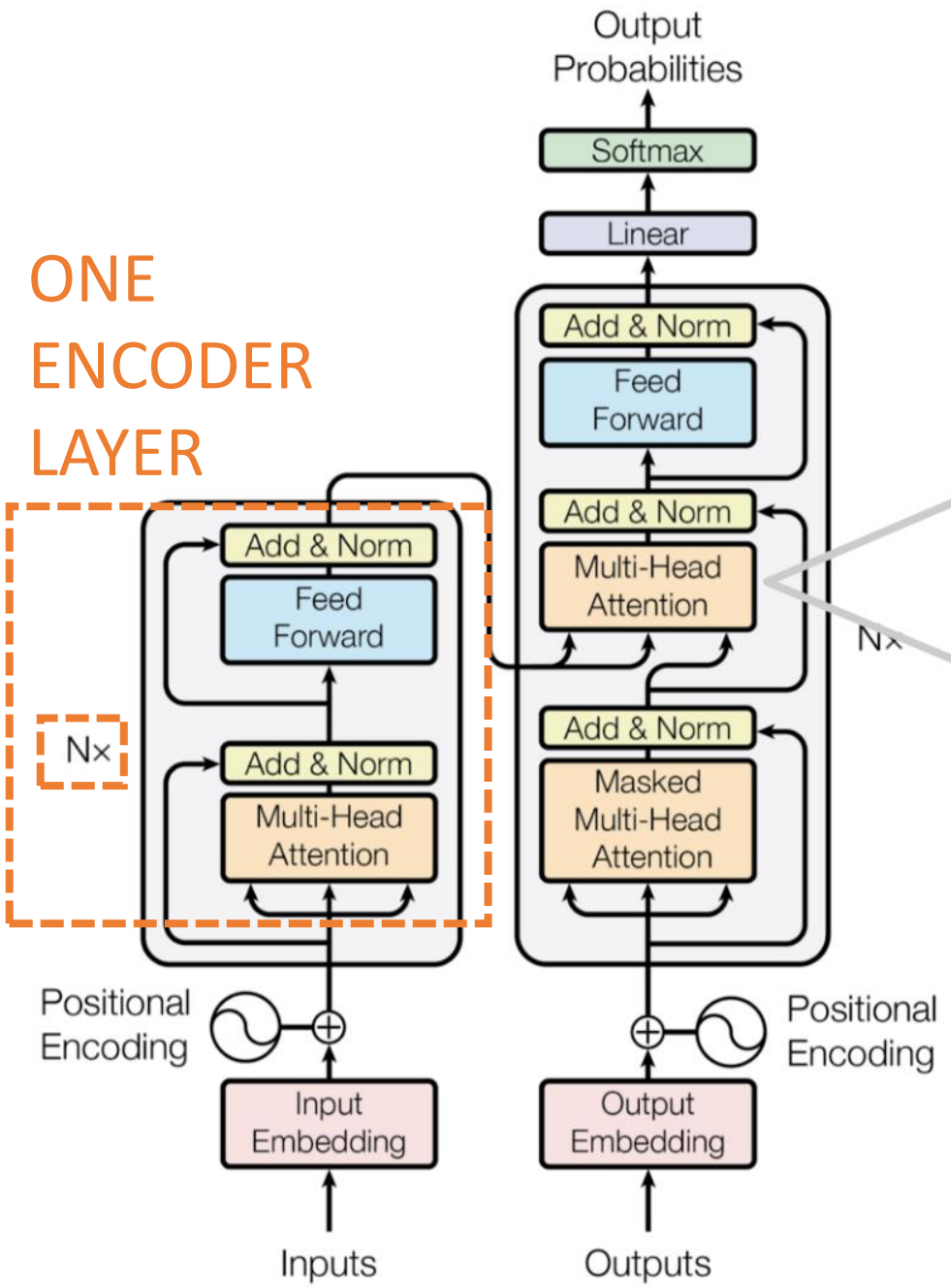
Definition (the multi-head self-attention layer a.k.a. the attention layer):

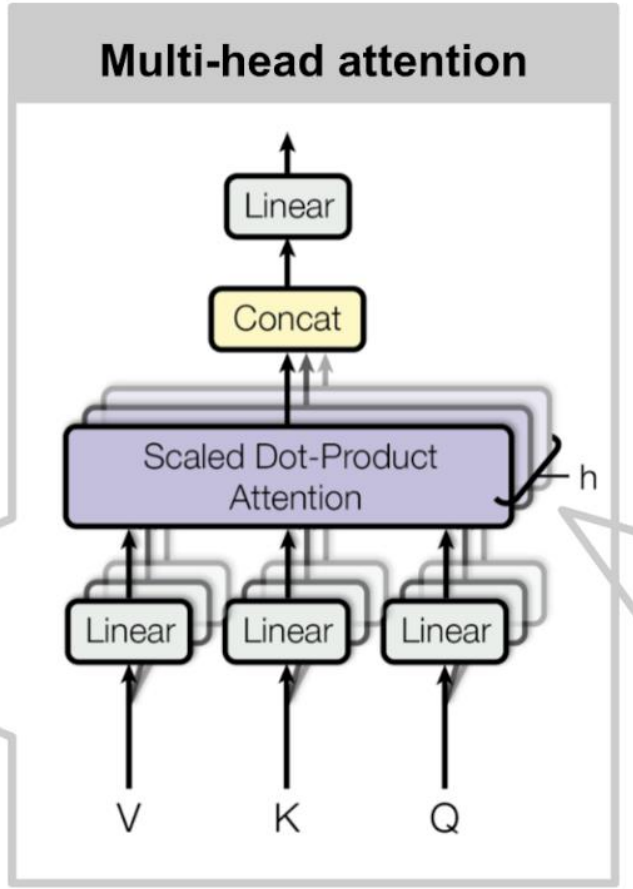
5. Feedforward is a set of linear/fully connected layers, whose main purpose is to reshape the attention vectors so that it is acceptable by the next encoder or decoder layer.

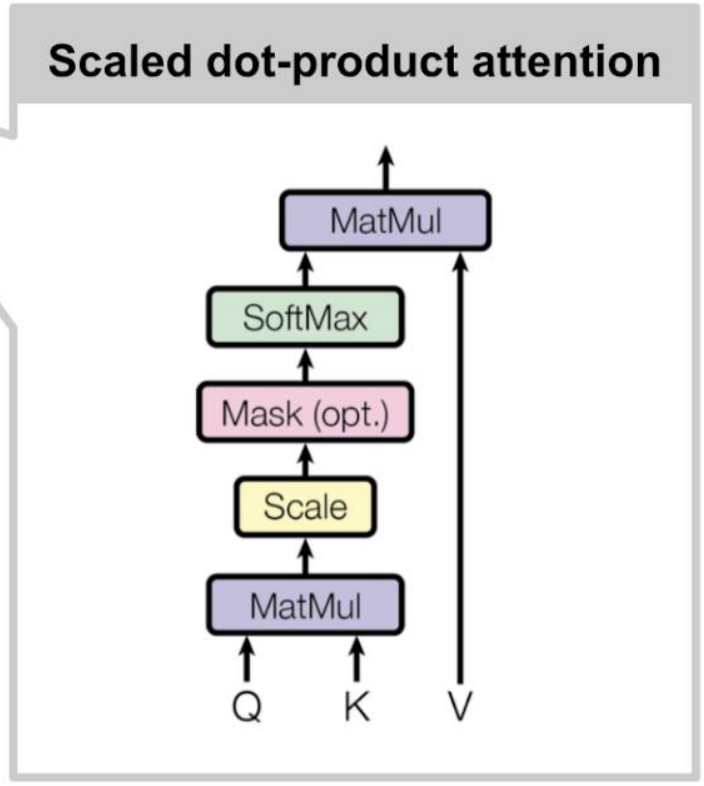
   Also uses a skip/residual connection as in ResNet!



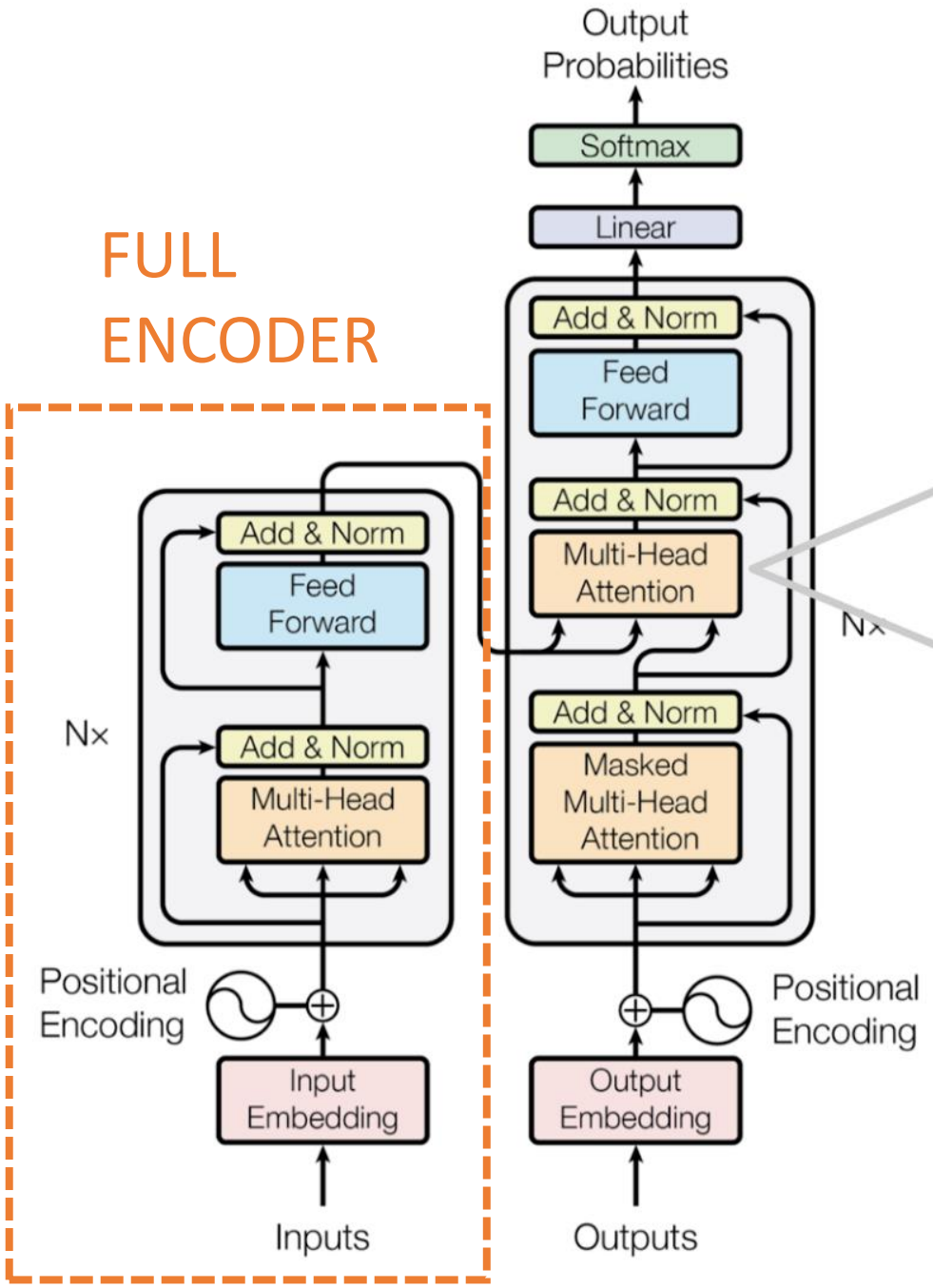
ONE ENCODER LAYER

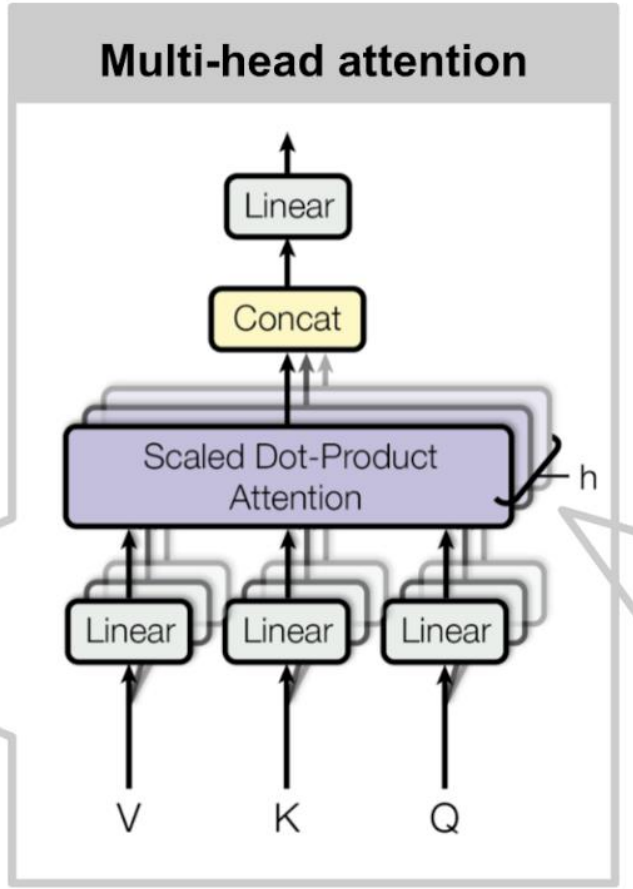


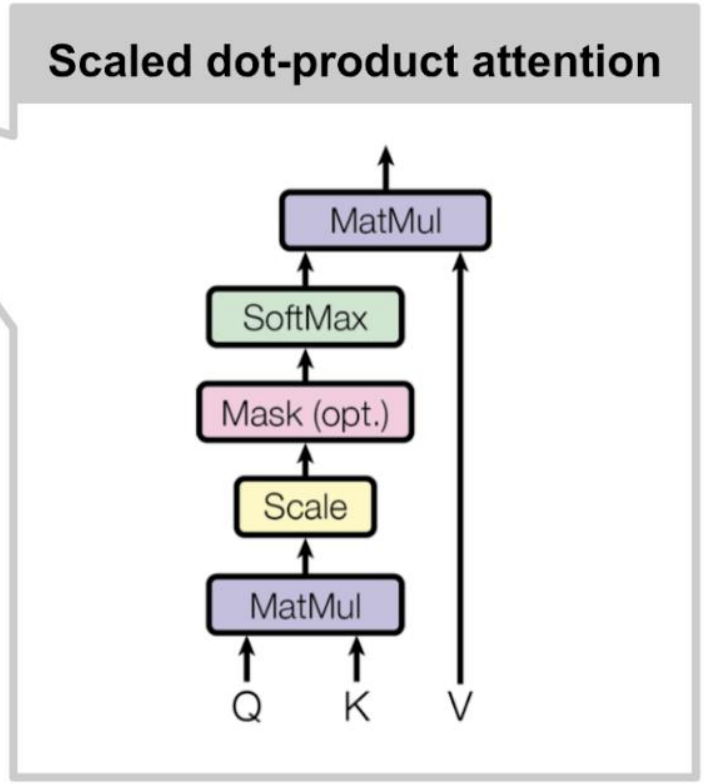




FULL ENCODER





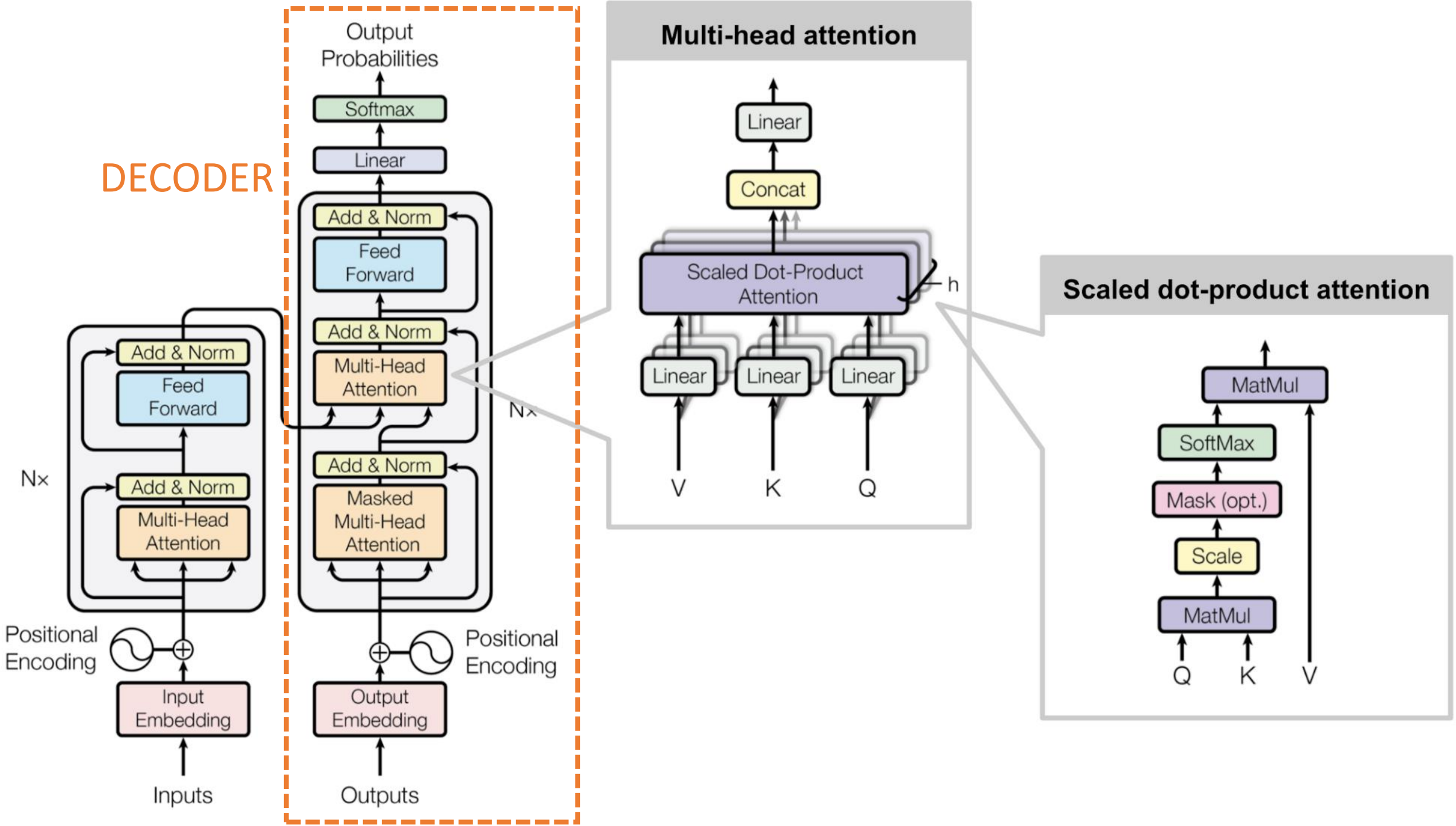


```
class EncoderLayer(nn.Module):
    """
    Description: EncoderLayer Class, describing the encoder layers to
    be used in the Encoder part of the transformer.
    Attributes list:
    - d_model: An integer, defining the dimension of the encoder layer.
    - num_heads: An integer, defining the number of attention heads
    to be used in both the encoder and decoder layers.
    - d_inner: An integer, defining the number of neurons in the Feed Forward layers.
    - droupout_rate: A float value between 0 and 1, corresponding to the Dropout rate
    used in the Dropout layers of the Feed Forward layers.
    """
```

```
    def __init__(self, d_model, num_heads, d_inner = 2048, droupout_rate = 0.3):
        """
        Init Method, which adds up
        - one MultiHeadAttentionLayer, with an expanded mask,
        - one Feed Forward layer,
        - and Normalization Layers after each one of them.
        """
        super().__init__()
        # First Multi Head Attention Layer
        self.attention = MultiHeadAttentionLayer(d_model, num_heads, droupout_rate)
        # Feed Forward Network Layer
        self.ffn = nn.Sequential(nn.Linear(d_model, d_inner),
                                 nn.ReLU(inplace = True),
                                 nn.Dropout(droupout_rate),
                                 nn.Linear(d_inner, d_model),
                                 nn.Dropout(droupout_rate),)
        # Normalization Layers
        # (Roughly identical to BatchNorm)
        self.attention_normalization = nn.LayerNorm(d_model)
        self.ffn_normalization = nn.LayerNorm(d_model)
```

```
    def forward(self, x):
        """
        Forward pass for the encoder layer.
        """
        # Attention Layer forward pass and normalization
        out = x + self.attention(q = x,
                                 k = x,
                                 v = x)
        out = self.attention_normalization(out)
        # Feed Forward Network Layer forward pass and normalization
        out = out + self.ffn(out)
        out = self.ffn_normalization(out)
        return out
```

```python
class Encoder(nn.Module):
    """
    Description: Encoder Class, describing the Encoding part of the transformer.
    Attributes list:
    - d_model: An integer, defining the dimension of the encoder layers.
    - num_heads: An integer, defining the number of attention heads
    to be used in both the encoder and decoder layers.
    - num_encoders: An integer, defining the number of encoder layers
    to be used in the transformer.
    """

    def __init__(self, d_model, num_heads, num_encoders):
        """
        Init Method, which adds up num_encoders EncoderLayers in a row.
        """
        super().__init__()
        layers_list = [EncoderLayer(d_model, num_heads) for i in range(num_encoders)]
        self.enc_layers = nn.ModuleList(layers_list, )

    def forward(self, x):
        """
        Forward pass for the Encoder, simply repeating the EncoderLayers in a row.
        """
        out = x
        for layers in self.enc_layers:
            out = layer(out)
        return out
```


DECODER
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
Multi-head attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q
Scaled dot-product attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V

The attention layer

Definition (the multi-head self-attention layer a.k.a. the attention layer):

6. Same multi-head block as before but with **masking** this time!

   Why masking?

While generating target sequences at the decoder, since the transformer uses self-attention, it tends to include all the words from the decoder inputs.

But, practically this is incorrect and "cheating" for the network.

The transformer must not use the current or future output to predict the next one, so the output sequence must be partially masked to prevent this reverse information flow.

Masked Multi-Head Attention ensures this.

The attention layer

Defi

atte

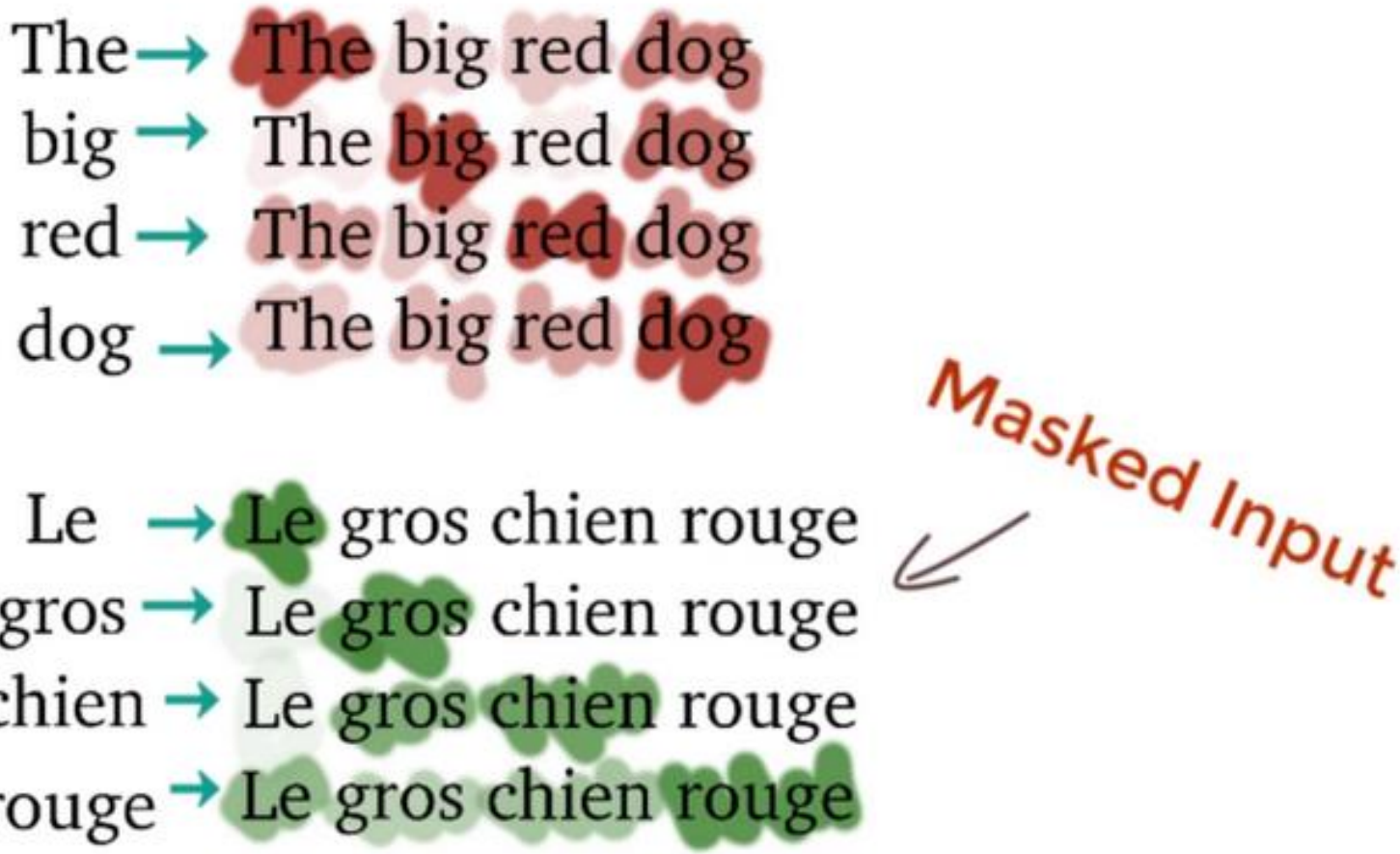
atte

6. S
   b
   t

   V

Encoder

Decoder



Masked Input

es at
er
clude

Only
word
n of

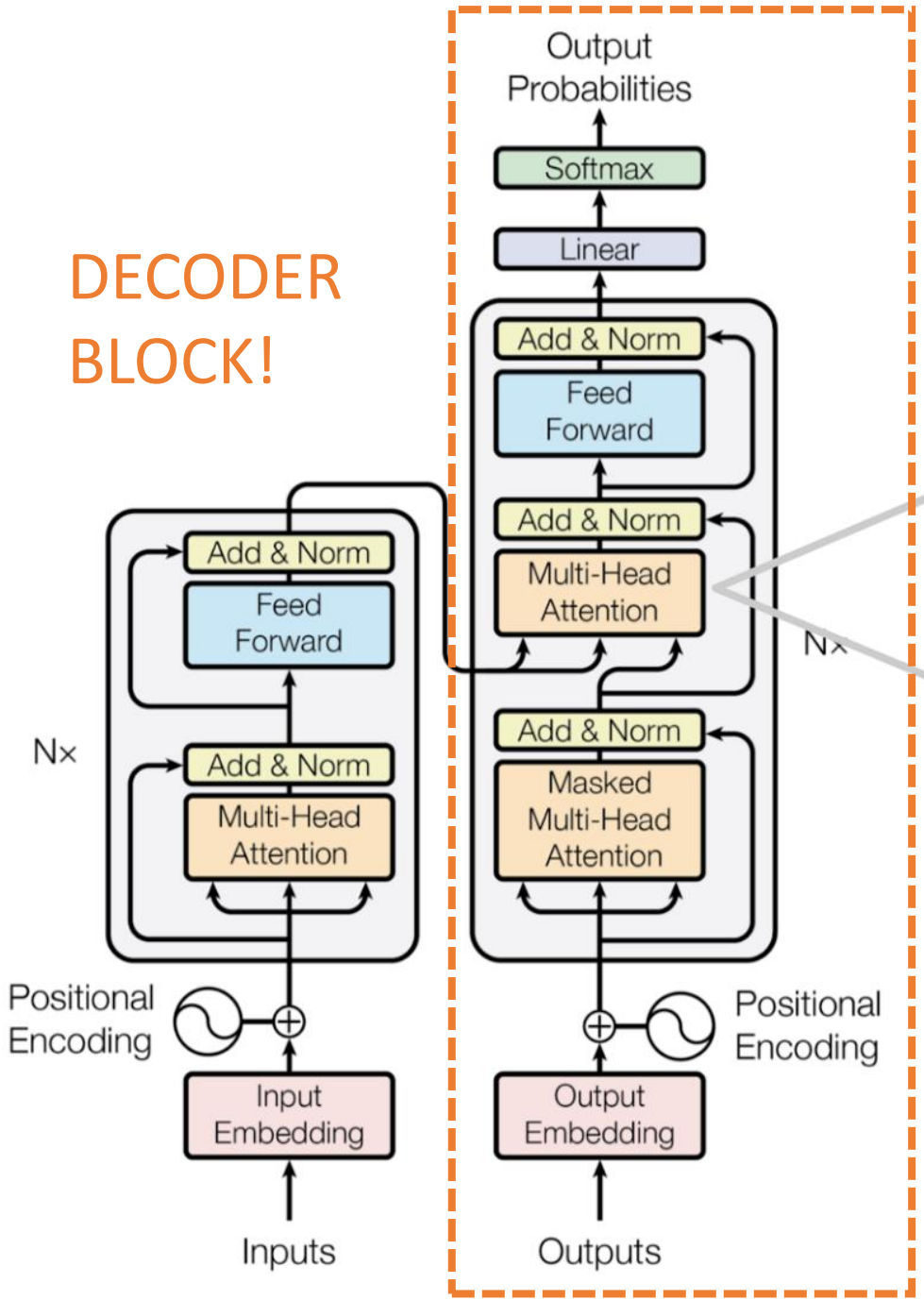
sures

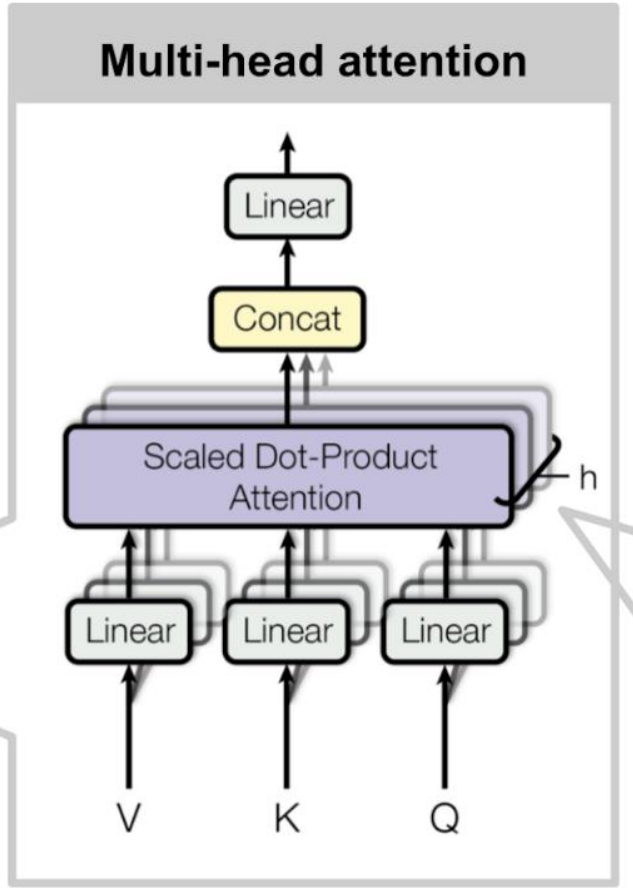
(Extra step in the attention layer class, forward method!)

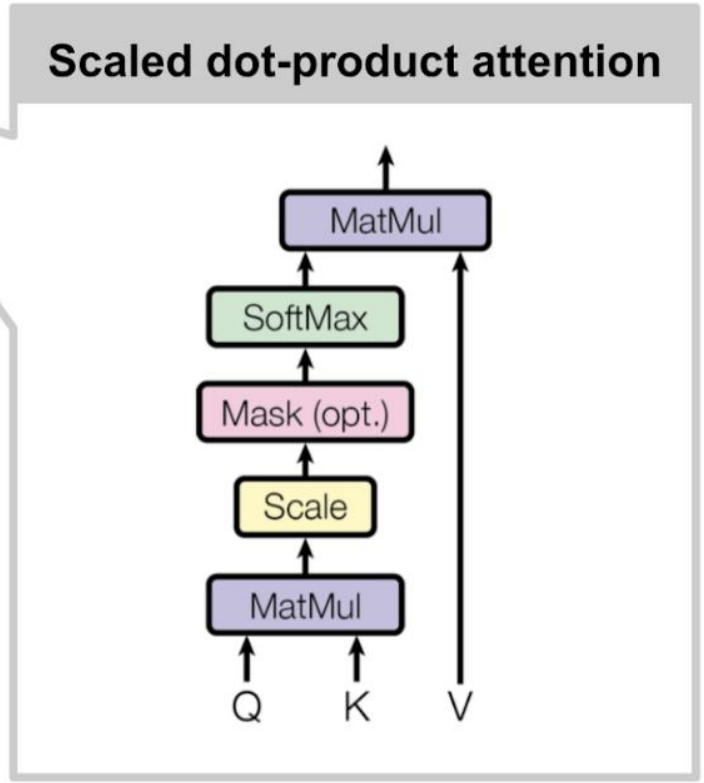
Note: I played around with a few different masks, just for fun. Paper suggests to use the triu (upper triangular) mask.

```
42
43          # Apply masks if needed
44          if self.mask is None:
45              continue
46          elif not self.mask:
47              temp = torch.ones((y_len, y_len), \
48                                device = self.mask.device, \
49                                dtype = torch.uint8)
50              subsequent_mask = 1 - torch.triu(temp, diagonal = 1)
51              subsequent_mask = subsequent_mask[None, :, :].expand(q_shape, y_len, y_len)
52              scores = scores.masked_fill(subsequent_mask == 0, -float("Inf"))
53          elif self.mask:
54              expanded_mask = self.mask[:, None, :].expand(q_shape, y_len, seq_len)
55              scores = scores.masked_fill(expanded_mask == 0, -float("Inf"))
56
```

DECODER BLOCK!





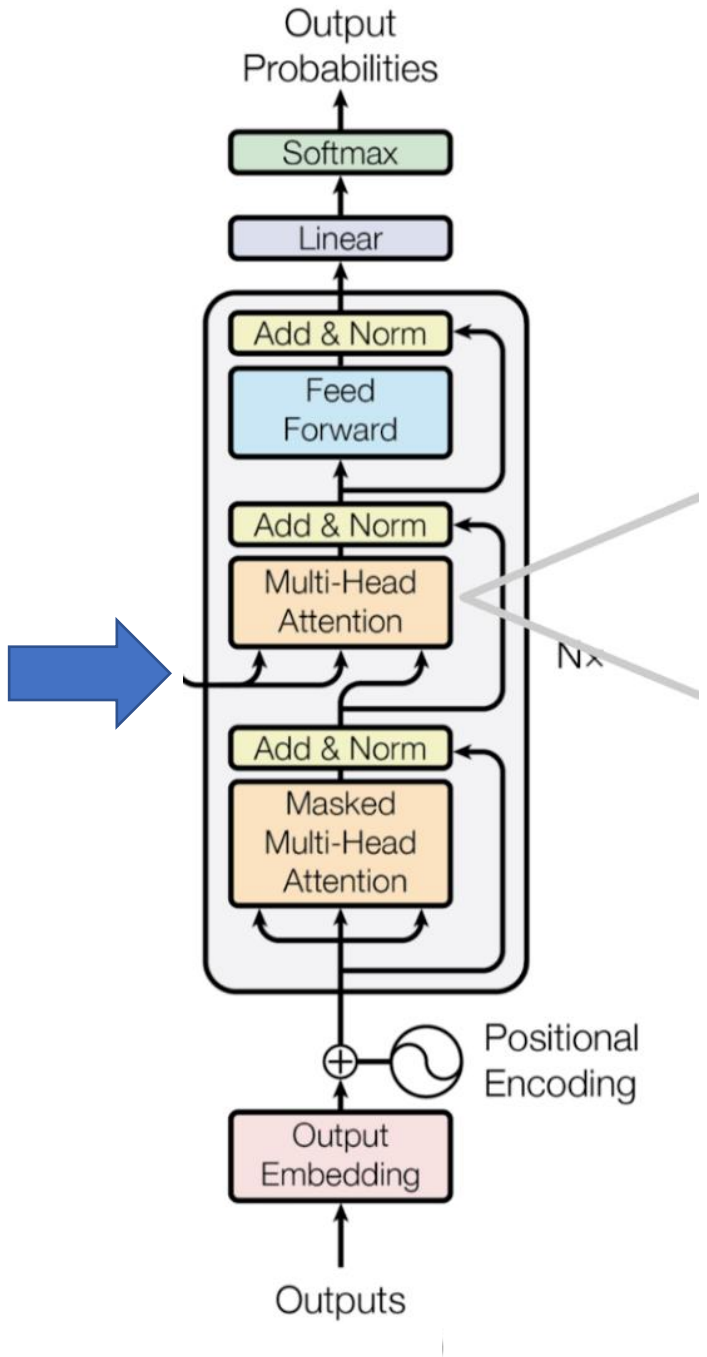


The attention layer

Definition (the multi-head self-attention layer a.k.a. the attention layer):

7. The second attention block is special as it uses V and K from the encoder part and Q from the decoder part.

$$A = softmax\left(\frac{QK^T}{d}\right)V$$

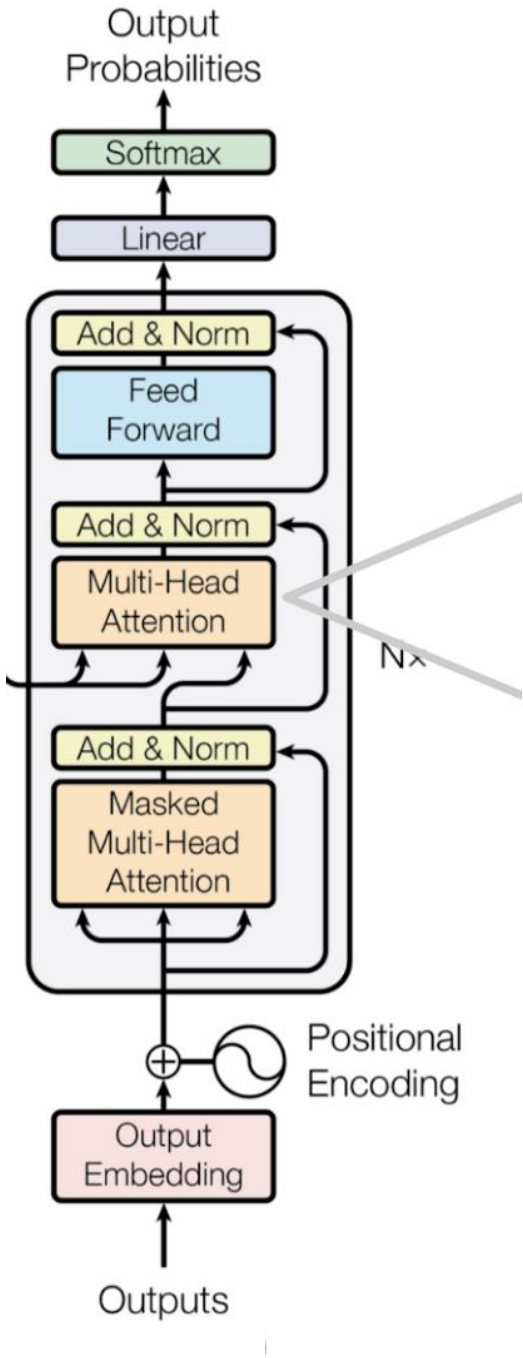


The attention layer

Definition (the multi-head self-attention layer a.k.a. the attention layer):

8. Repeat the attention block (multi-head attention + add/norm + feed forward) a few times.

   Finish with a fully connected and softmax giving probabilities for the words in each position of the output sequence!



```python
class DecoderLayer(nn.Module):
    """
    Description: DecoderLayer Class, describing the decoder layers to
    be used in the Decoder part of the transformer.
    Attributes list:
    - d_model: An integer, defining the dimension of the encoder layers.
    - num_heads: An integer, defining the number of attention heads
    to be used in the decoder layer.
    - d_inner: An integer, defining the number of neurons in the Feed Forward layers.
    - droupout_rate: A float value between 0 and 1, corresponding to the Dropout rate
    used in the Dropout layers of the Feed Forward layers.
    """
```

```
    def __init__(self, d_model, num_heads, d_inner = 2048, droupout_rate = 0.3):
        """
        Init Method, which adds up
        - one MultiHeadAttentionLayer, with a triangular mask,
        - one MultiHeadAttentionLayer, with no mask,
        - one Feed Forward layer,
        - and Normalization Layers after each one of them.
        """
        super().__init__()
        # Masked First Attention Layer
        self.masked_attention = MultiHeadAttentionLayer(d_model, \
                                                        num_heads, \
                                                        droupout_rate, \
                                                        mask = True)
        # Subsequent Attention Layer
        self.subsequent_attention = MultiHeadAttentionLayer(d_model, \
                                                            num_heads, \
                                                            droupout_rate)
        # Feed Forward Network Layer
        self.ffn = nn.Sequential(nn.Linear(d_model, d_inner),
                                 nn.ReLU(inplace = True),
                                 nn.Dropout(droupout_rate),
                                 nn.Linear(d_inner, d_model),
                                 nn.Dropout(droupout_rate),)
        # Normalization Layers
        # (Roughly identical to BatchNorm)
        self.masked_attention_normalization = nn.LayerNorm(d_model)
        self.subsequent_attention_normalization = nn.LayerNorm(d_model)
        self.ffn_normalization = nn.LayerNorm(d_model)
```

```
    def forward(self, y, x):
        """
        Forward pass for the decoder layer.
        """
        # Masked First Attention Layer
        out = y
        out = out + self.masked_attention(q = out, k = out, v = out)
        out = self.masked_attention_normalization(out)
        # Subsequent Attention Layer
        out = out + self.subsequent_attention(q = out, k = x, v = x)
        out = self.subsequent_attention_normalization(out)
        # Feed Forward Network Layer
        out = out + self.ffn(out)
        out = self.ffn_normalization(out)
        return out
```

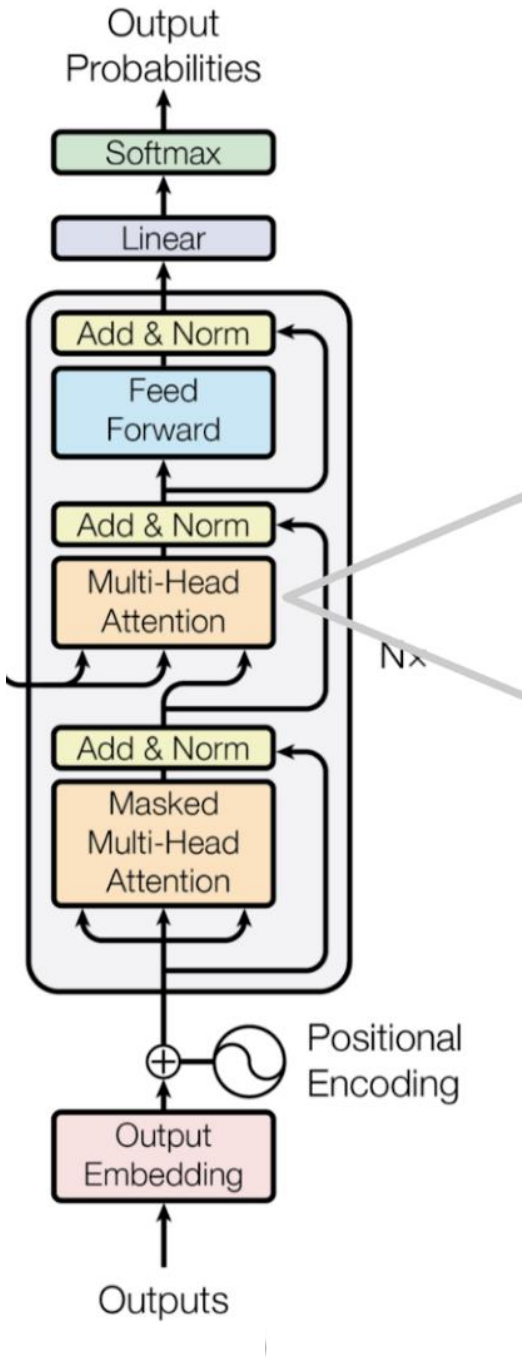
```python
class Decoder(nn.Module):
    """
    Description: Encoder Class, describing the Encoding part of the transformer.
    Attributes list:
    - d_model: An integer, defining the dimension of the decoder layers.
    - num_heads: An integer, defining the number of attention heads
    to be used in both the decoder layers.
    - num_decoders: An integer, defining the number of decoder layers
    to be used in the transformer.
    """

    def __init__(self, d_model, num_heads, num_decoders):
        """
        Init Method, which adds up num_decoders DecoderLayers in a row.
        """
        super().__init__()
        layers_list = [DecoderLayer(d_model, num_heads) for i in range(num_decoders)]
        self.dec_layers = nn.ModuleList(layers_list, )

    def forward(self, y, enc):
        """
        Forward pass for the Decoder, simply repeating the DeccoderLayers in a row.
        """
        out = y
        for layers in self.dec_layers:
            out = layer(out, enc)
        return out
```

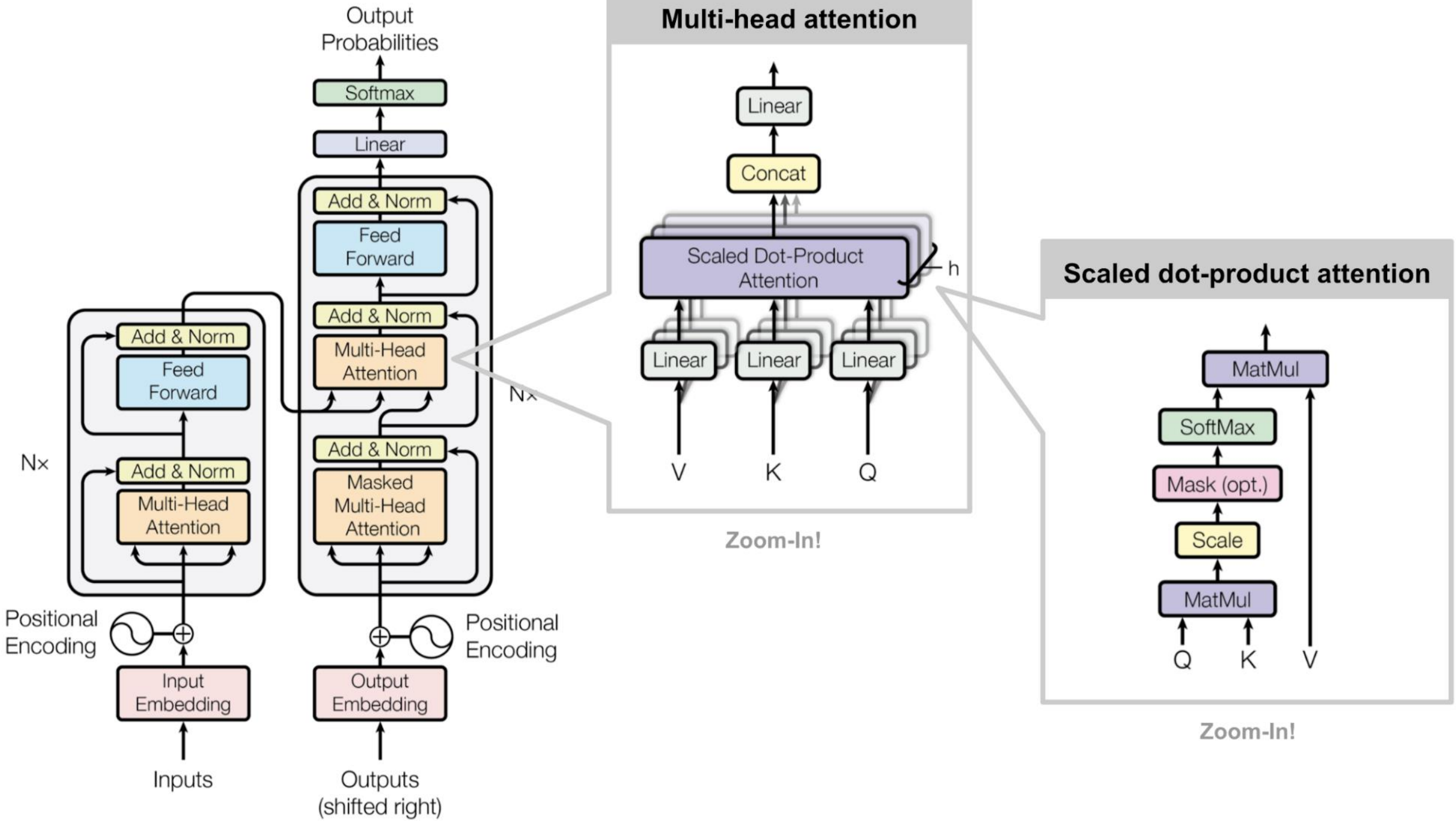
The attention layer

Definition (the multi-head self-attention layer a.k.a. the attention layer):

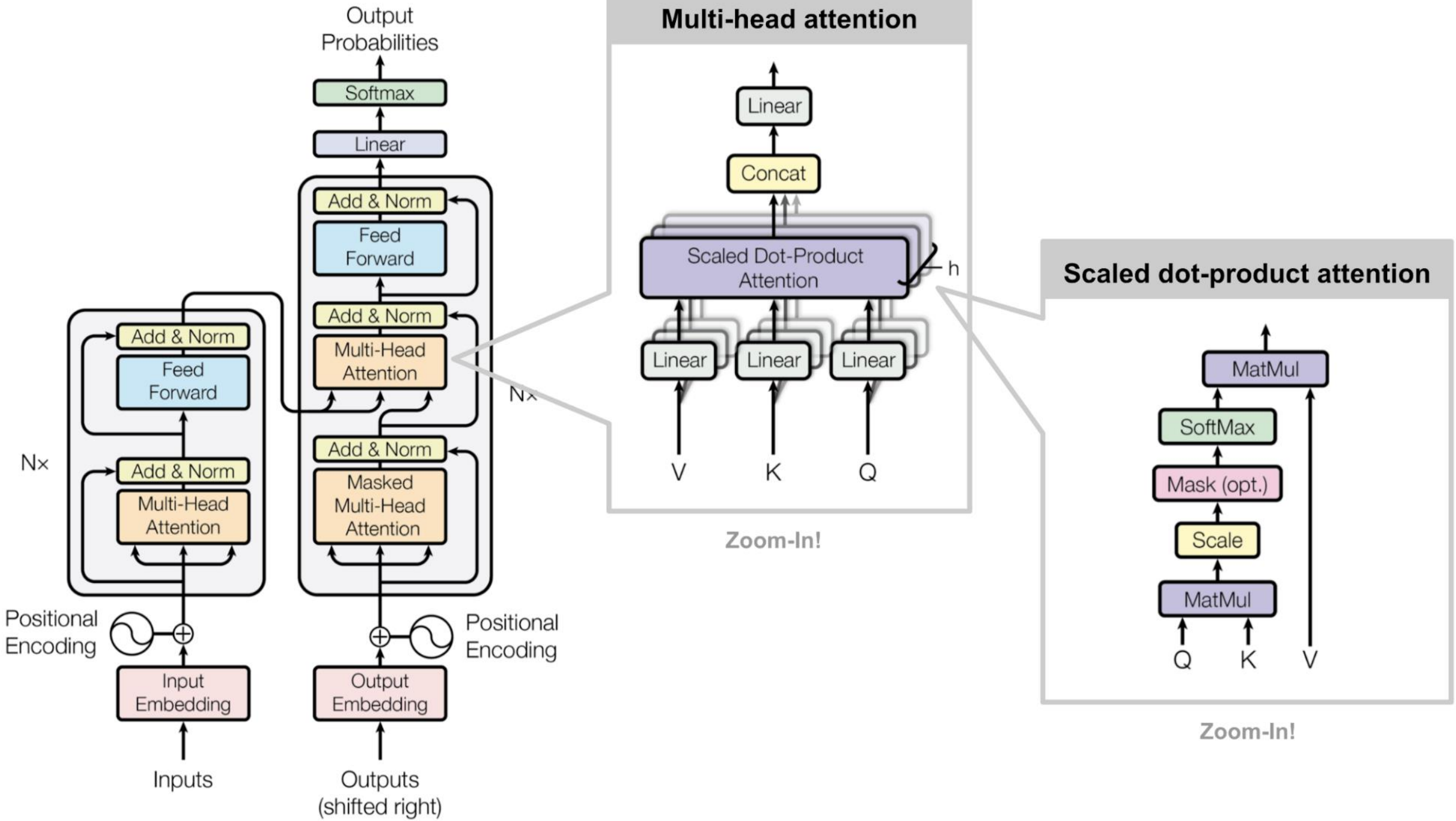
8. Train it with several pairs (x, y) of sentences in two different languages.

   Can be approached as a classification task, so no shenanigans on the loss and optimizer parts, as usual.

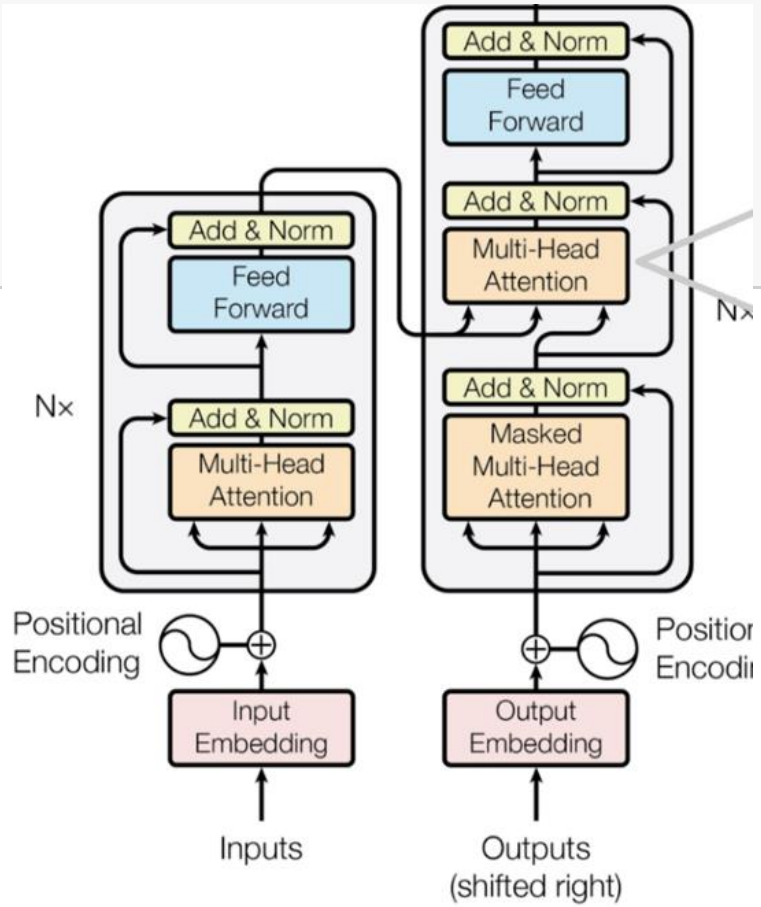



Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
Multi-head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

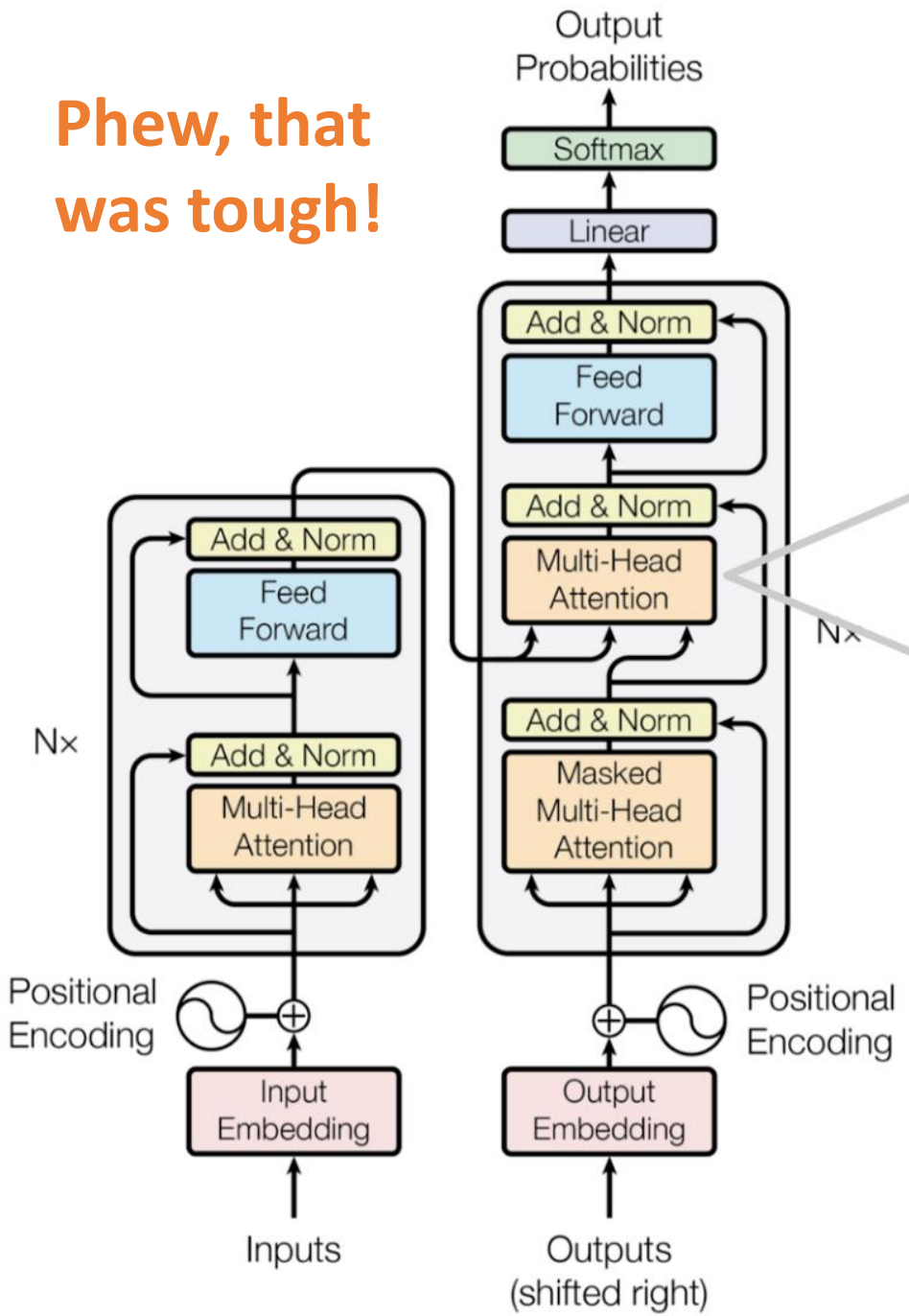
```python
class Transformer(nn.Module):
    """
    Description: Full Transformer Class, combining the Encoder and Decoder parts.
    Attributes list:
    - d_model: An integer, defining the dimension of the encoder
    and decoder layers.
    - num_heads: An integer, defining the number of attention heads
    to be used in both the encoder and decoder layers.
    - num_encoders: An integer, defining the number of encoder layers
    to be used in the transformer.
    - num_decoders: An integer, defining the number of decoder layers
    to be used in the transformer.
    """

    def __init__(self, d_model = 512, num_heads = 8, num_encoders = 6, num_decoders = 6):
        """
        Init Method, establishing the Decoder and Encoder parts as attributes of the transfo
        Later on, we will extract the encoder part for our Embeddings.
        """
        super().__init__()
        self.encoder = Encoder(d_model, num_heads, num_encoders)
        self.decoder = Decoder(d_model, num_heads, num_decoders)
```

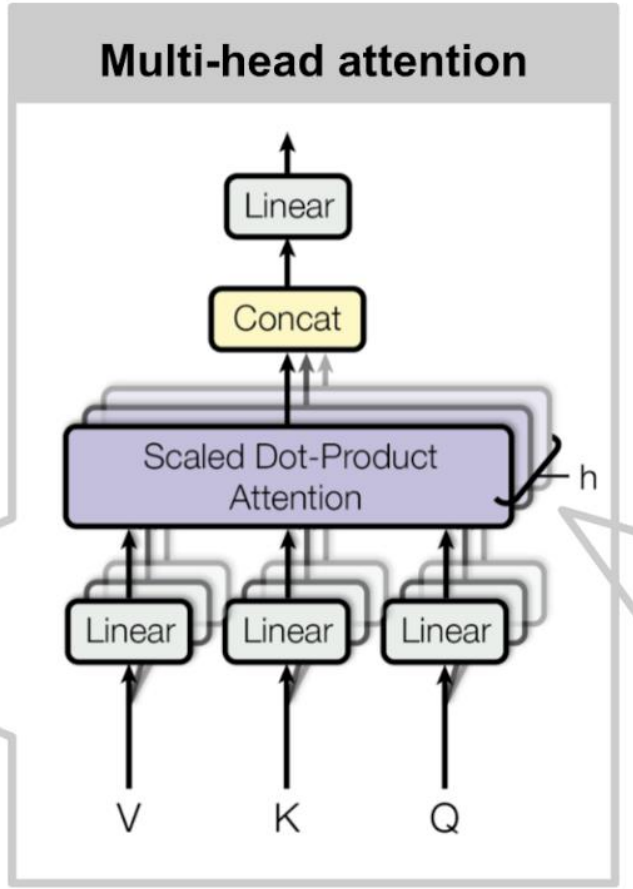

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
Multi-head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

```
    def forward(self, x, y):
        """
        Forward pass for transformer.
        - Will encoder the input with the Encoder part first.
        - Then use the Decoder, combining the encoded input and its target,
        along with masks.
        """
        enc_out = self.encoder(x)
        dec_out = self.decoder(y, enc_out)
        return dec_out
```

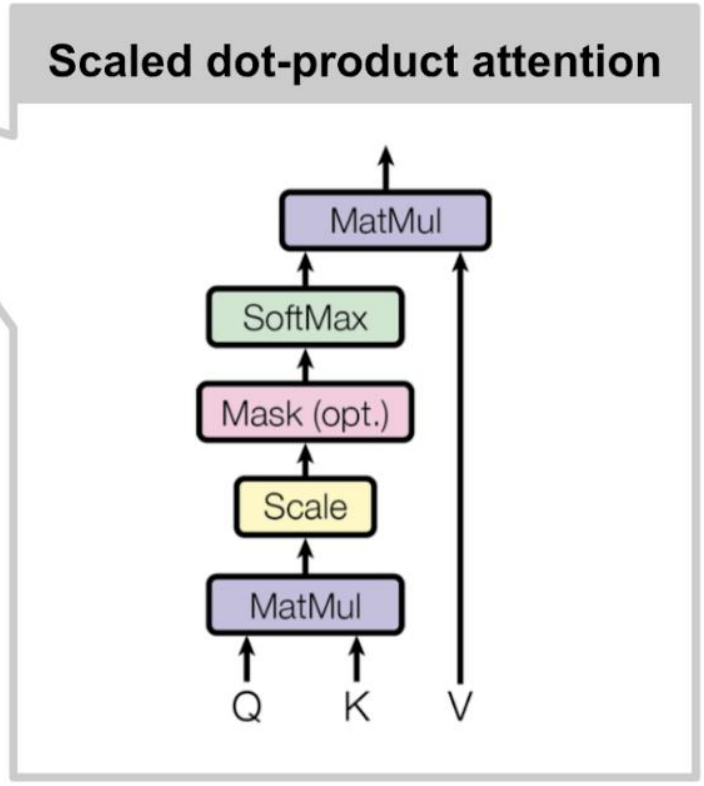


Phew, that was tough!



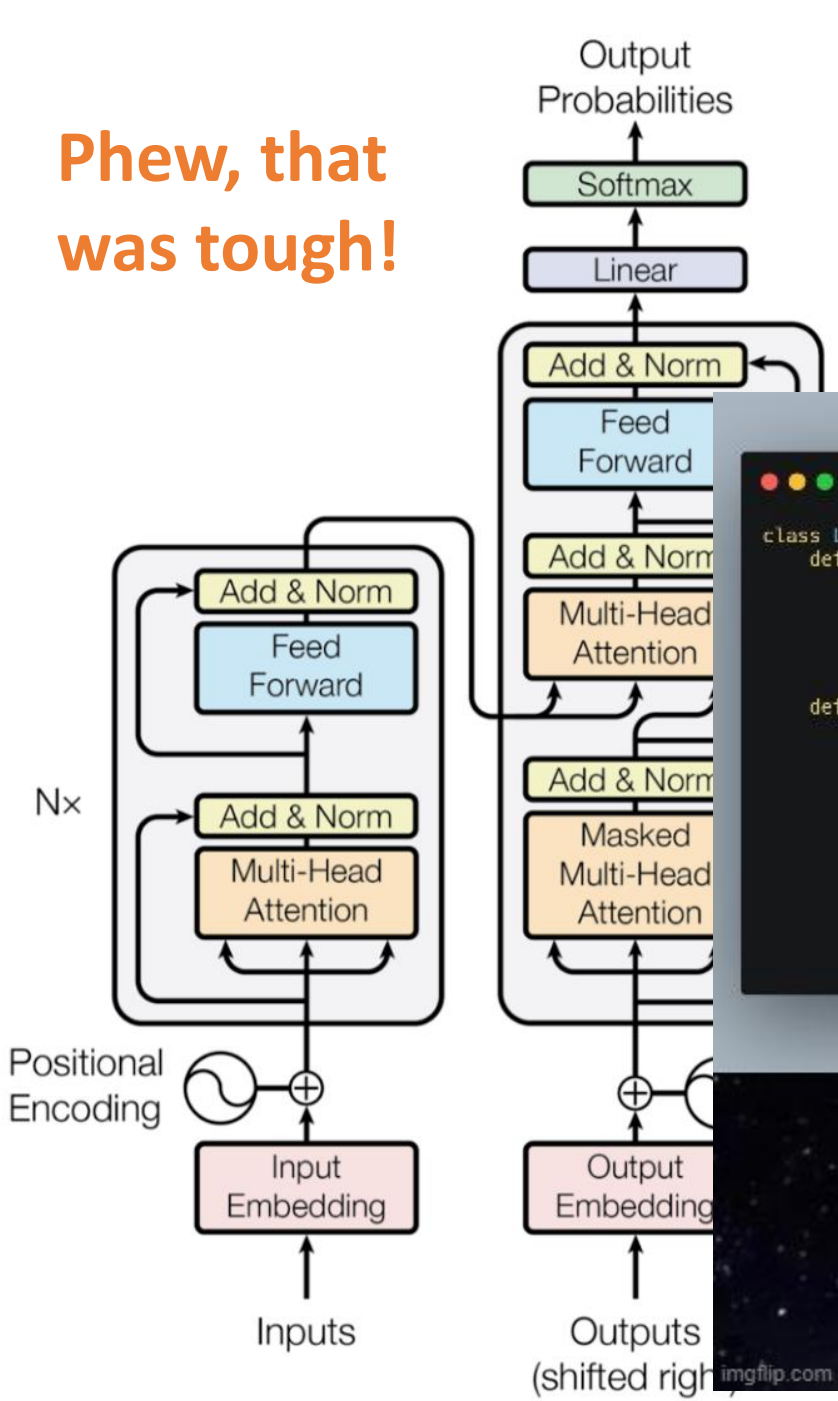


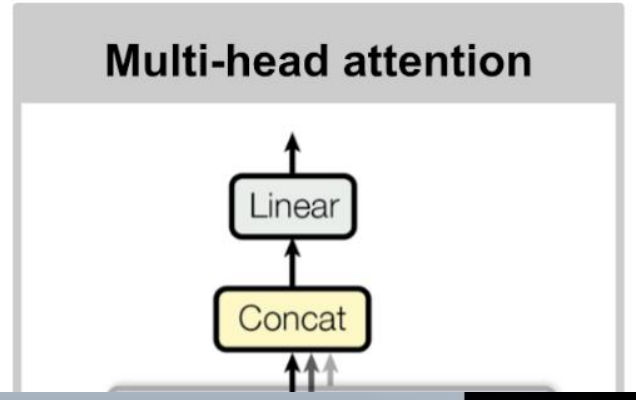
Zoom-In!



Zoom-In!

Phew, that was tough!





Credit fast.ai

```
class LSTMCell(Module):
    def __init__(self, ni, nh):
        self.forget_gate = nn.Linear(ni + nh, nh)
        self.input_gate  = nn.Linear(ni + nh, nh)
        self.cell_gate   = nn.Linear(ni + nh, nh)
        self.output_gate = nn.Linear(ni + nh, nh)

    def forward(self, input, state):
        h,c = state
        h = torch.stack([h, input], dim=1)
        forget = torch.sigmoid(self.forget_gate(h))
        c = c * forget
        inp = torch.sigmoid(self.input_gate(h))
        cell = torch.tanh(self.cell_gate(h))
        c = c + inp * cell
        out = torch.sigmoid(self.output_gate(h))
        h = outgate * torch.tanh(c)
        return h, (h,c)
```

Great news!

A recent release of PyTorch **COMES WITH TRANSFORMERS CLASSES!**

https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html

TRANSFORMER

CLASS `torch.nn.Transformer(d_model=512, nhead=8, num_encoder_layers=6, num_decoder_layers=6, dim_feedforward=2048, dropout=0.1, activation=<function relu>, custom_encoder=None, custom_decoder=None, layer_norm_eps=1e-05, batch_first=False, norm_first=False, device=None, dtype=None)` [SOURCE]

A transformer model. User is able to modify the attributes as needed. The architecture is based on the paper "Attention Is All You Need". Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems, pages 6000-6010. Users can build the BERT(https://arxiv.org/abs/1810.04805) model with corresponding parameters.

Applications for attention/transformers

- **In NLP:** Sentences are typical examples of sequences where computing attention coefficients make sense as a way to help with contextualization.

- Other applications for attention exist, typically with images, where attention coefficients can help understand how pixels interact with each other! Technically attention layers could “replace” CNNs!
- Or how pixels could affect a caption decision (image and video captioning, handwritten recognition, etc.)
- Will also be used in graphs next week!

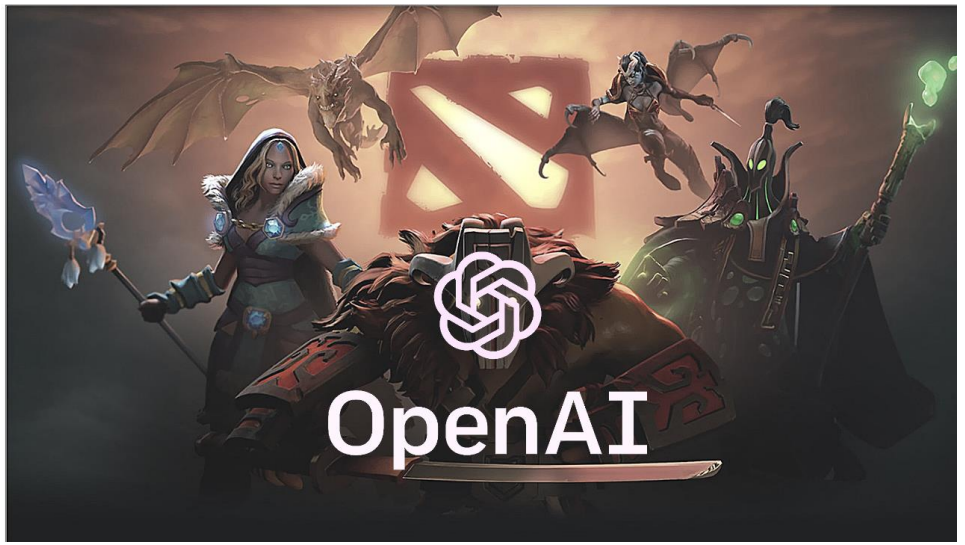
The BERT word embedding

- Why discuss transformers during this (long) lecture on embeddings?
- Well, let us assume we train a transformer for a meaningless linguistics task…
- Can we extract the encoder part, as we did with all the unsupervised approaches to word embedding (CBoW, SkipGram, FastText, ELMo)?
- **Yes! And you obtain the (current best?) word embedding, BERT!**

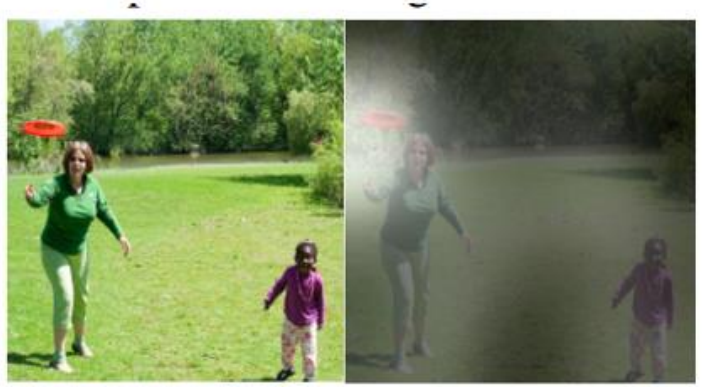
- Learn more about it in [Devlin2019]!
- Python package: https://pypi.org/project/bert-embedding/

Also, GPT-n embeddings

- On another note, OpenAI has produced an important embedding, also relying on transformers, called GPT-3 [Brown2020].
- This is also considered one of the state-of-the-art embeddings.
- **Fun fact:** the OpenAI company is also famous for AI playing video games, they are for instance behind the DoTA2 AI! See [Berner2020].



Out-of-NLP applications for transformers: Image captioning



A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

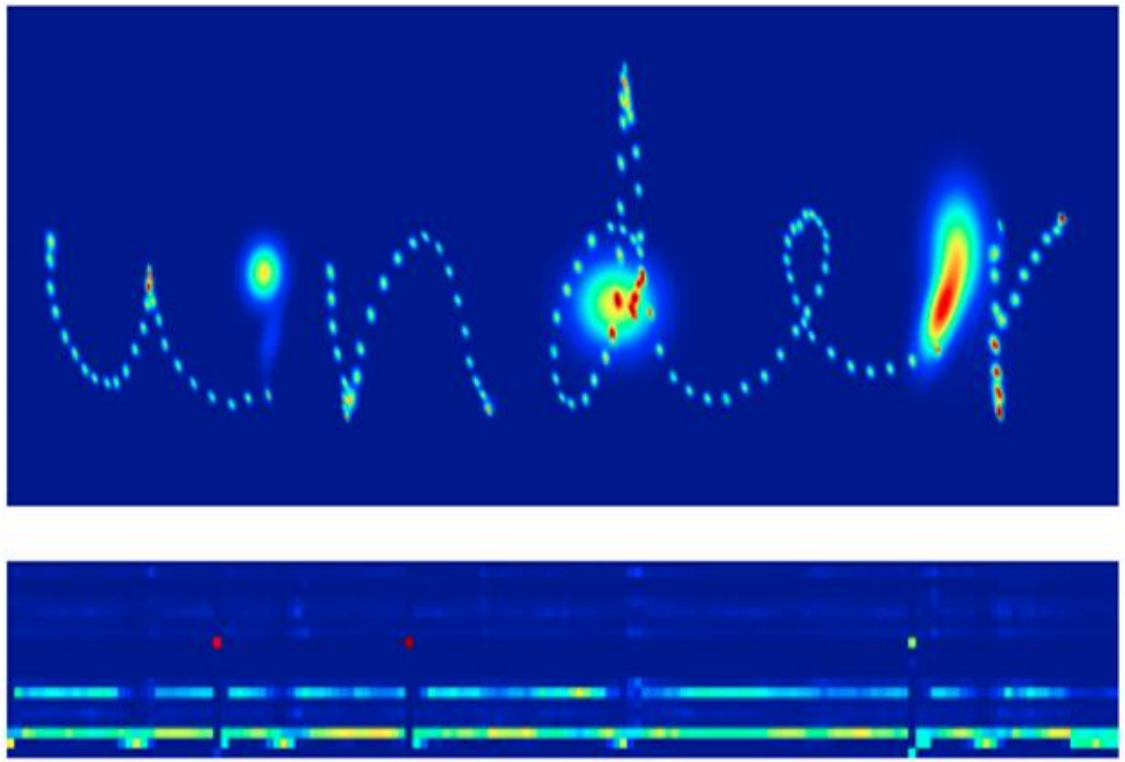
A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Out-of-NLP applications for transformers: Video captioning

Out-of-NLP applications for transformers: Handrwitten recognition



more of national temperement
more of national temperament
more of national temperament
more of national temperament
more of national temperament
more of national temperament

On the limits of attention?

- Active research following the BERT embedding ([Devlin2019])
- In the paper [Dong2021], a research team from Google and EPFL proposed a novel approach that sheds light on the operation and inductive biases of self-attention/transformer networks and finds that pure attention performance decays in rank doubly exponentially with respect to depth.
- Recent works suggest that a new type of transformer layer, called **Lambda layers** could become the next hype for the future of NLP embeddings [Bello2021]?
- To be confirmed!

Conclusion (W9S3)

We have seen a few approaches to embeddings.

- Train an AI to figure out embeddings?
  - Basic approaches.
- Out of dictionary entries?
  - Use ELMo or FastText!

Advanced methods

- Supervised: InferSent
- Multi-task: Google's Universal Embedding.

A few more problems are still open at the moment for these word embeddings.

- Can these embedding be biased? Yes, unfortunately, active research on this.
- Can we help the network in computing context? Yes, with attention and transformers!
- Best word embedding at the moment? Probably BERT or a variation of BERT or GPT-3!

Learn more about these topics

Out of class, for those of you who are curious

- [Schnabel2015] **Schnabel** et al., “Evaluation methods for unsupervised word embeddings”, 2015. https://www.aclweb.org/anthology/D15-1036/
- [Bolukbasi2016] **Bolukbasi** et al., “Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings”, 2016. https://arxiv.org/abs/1607.06520
- [Vaswani2017] **Vaswani** et al., “Attention Is All You Need”, 2017. https://arxiv.org/abs/1706.03762

Learn more about these topics

Out of class, for those of you who are curious

- [Conneau2018] **Conneau** et al., “Supervised Learning of Universal Sentence Representations from Natural Language Inference Data”, 2018. https://arxiv.org/abs/1705.02364
- [Cer2019] **Cer** et al., “Universal Sentence Encoder”, 2019. https://arxiv.org/abs/1803.11175
- [Wang2018] Wang et al., “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding” https://arxiv.org/abs/1804.07461

Learn more about these topics

- [Devlin2019] **Devlin** et al., “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, 2019. https://arxiv.org/abs/1810.04805
- [Dong2021] Dong et al., “Attention is Not All You Need: Pure Attention Loses Rank Doubly Exponentially with Depth”, 2021. https://arxiv.org/abs/2103.03404
- [Bello2021] Bello et al., “LambdaNetworks: Modeling Long-Range Interactions Without Attention”, 2021. https://arxiv.org/abs/2102.08602

Learn more about these topics

- [Berner2020] Berner et al., “Dota 2 with Large Scale Deep Reinforcement Learning”, 2020. https://arxiv.org/abs/1912.06680 https://www.vox.com/2019/4/13/18309418/open-ai-dota-triumph-og
- [Brown2020] Brown et al. (20 of them or so), “Language Models are Few-Shot Learners”, 2020. https://arxiv.org/abs/2005.14165
- Read more about GPT-3and OpenAI, here: https://www.vox.com/future-perfect/21355768/gpt-3-ai-openai-turing-test-language https://www.technologyreview.com/2020/07/20/1005454/openai-machine-learning-language-generator-gpt-3-nlp/

Learn more about these topics

Tracking important names (Track their works and follow them on Scholar, Twitter, or whatever works for you!)

- **Alexis Conneau:** Senior Researcher at **Google**.
  https://scholar.google.fr/citations?user=45KfCpgAAAAJ&hl=fr
- **Matthew E. Peters:** Senior researcher at **Allen Institute.**
  https://scholar.google.com/citations?user=K5nCPZwAAAAJ&hl=en
- **Daniel Cer:** Researcher at **Google** and Lecturer at **UC Berkeley.**
  Has a really good course about NLP and Embeddings.
  https://scholar.google.com/citations?user=BrT1NW8AAAAJ&hl=en
  https://www.ischool.berkeley.edu/user/4406
  https://www.ischool.berkeley.edu/courses/datasci/266

Learn more about these topics

Tracking important names (Track their works and follow them on Scholar, Twitter, or whatever works for you!)

- **Tobias Schnabel:** Senior researcher at **Microsoft**.
  https://scholar.google.com/citations?user=o_i419UAAAAJ&hl=en
  https://www.microsoft.com/en-us/research/people/toschnab
  Tolga Bolukbasi: Senior researcher at **Google Brain**.
  https://scholar.google.com/citations?user=3rF9gtAAAAAJ&hl=en
- **Anish Vaswani:** Researcher at **Google Brain**. Highest cited paper in DL.
  https://scholar.google.com/citations?user=oR9sCGYAAAAJ&hl=en
- **Jacob Devlin:** Researcher at **Google**.
  https://research.google/people/106320/

+ OpenAI!