IMPERIAL

Transformer

05/12/2025

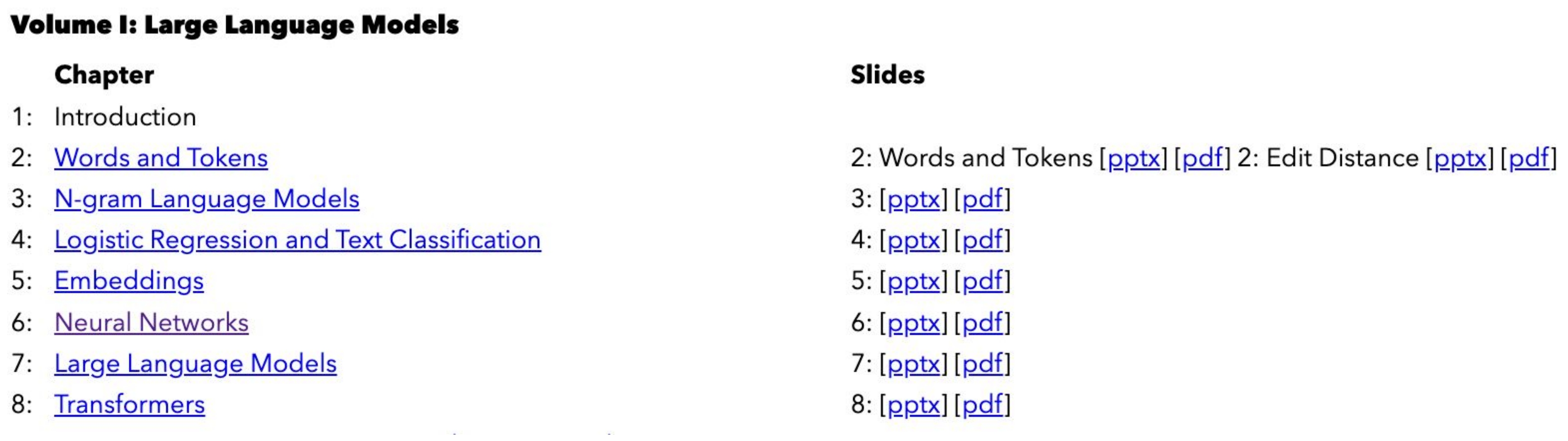
Shamsuddeen Muhammad
Google DeepMind Academic Fellow,
Imperial College London
https://shmuhammadd.github.io/

Idris Abdulmumin
Postdoctoral Research Fellow,
DSFSI, University of Pretoria
https://abumafrim.github.io/

Reference

These slides are based on the course material by Daniel Jurafsky :

Chapter 8: Transformers



Volume I: Large Language Models

	Chapter	**Slides**
1:	Introduction	
2:	Words and Tokens	2: Words and Tokens [pptx] [pdf] 2: Edit Distance [pptx] [pdf]
3:	N-gram Language Models	3: [pptx] [pdf]
4:	Logistic Regression and Text Classification	4: [pptx] [pdf]
5:	Embeddings	5: [pptx] [pdf]
6:	Neural Networks	6: [pptx] [pdf]
7:	Large Language Models	7: [pptx] [pdf]
8:	Transformers	8: [pptx] [pdf]

From Self-Attention to Transformers

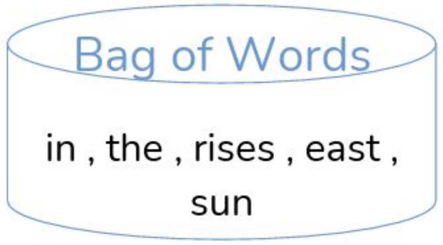
- We will talk about a class of models for processing sequences that does not use recurrent connections but instead relies entirely on attention and will build up towards a class of models called **Transformers**.
- To address a few key limitations, we need to add certain elements:

1. Positional encoding	addresses lack of sequence information
2. Multi-headed attention	allows querying multiple positions at each layer
3. Adding nonlinearities	so far, each successive layer is *linear* in the previous one
4. Masked decoding	how to prevent attention lookups into the future?

Positional Encoding - Motivation

- **Problem :** Self-attention processes all the elements of a sequence in parallel without any regard for their order.
  - Example : the sun rises in the east
  - Permuted version : rises in the sun the east

    the east rises in the sun
  - Self-attention is permutation invariant.
  - In natural language, it is important to take into account the order of words in a sentence.
- **Solution :** Explicitly add positional information to indicate where a word appears in a sequence



Transformers process input as sets of tokens — they are **order-invariant**.

Unlike RNNs or CNNs, Transformers lack built-in mechanisms to capture word order.

Positional encoding injects order information into token embeddings.

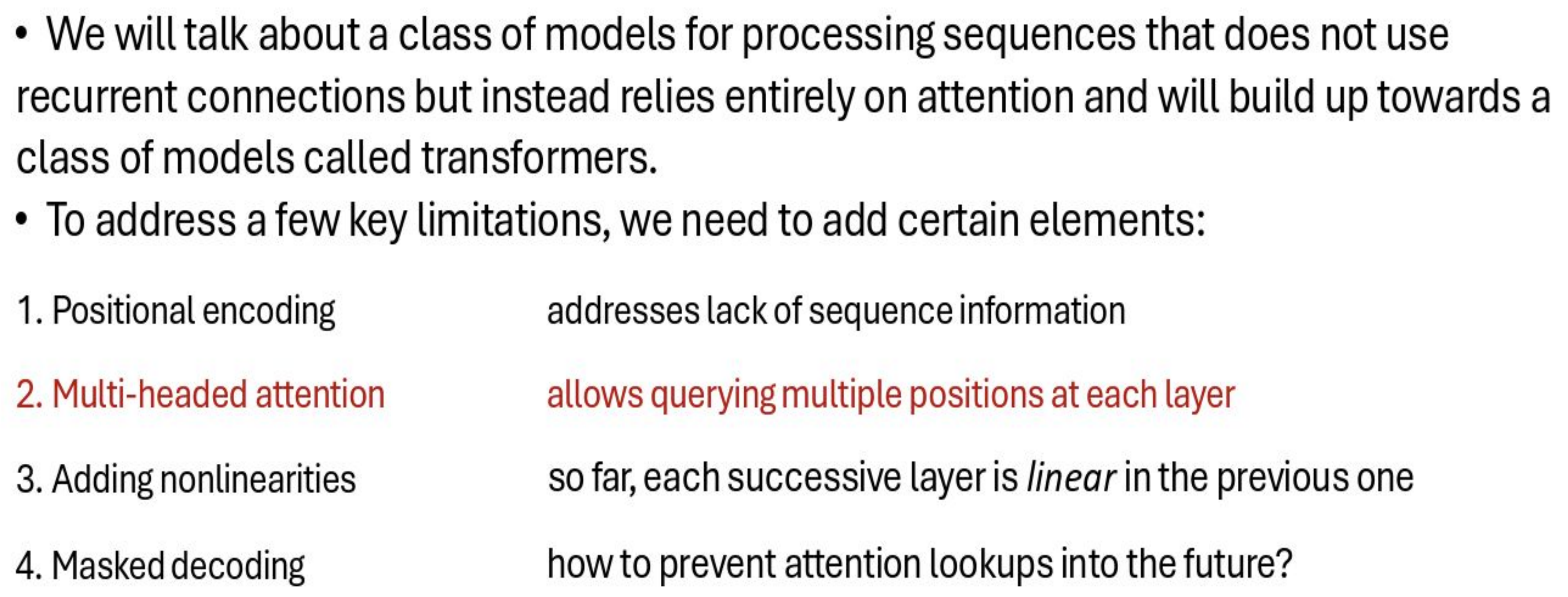
Positional encoding

Positional encoding is a mechanism to inject information about the order of tokens in a sequence.

Since Transformers lack recurrence (RNNs), they treat input sequences as unordered sets.

To enable the model to capture **word order and relative positions**, positional encodings are added to the input embeddings.

From Self-Attention to Transformers



- We will talk about a class of models for processing sequences that does not use recurrent connections but instead relies entirely on attention and will build up towards a class of models called transformers.
- To address a few key limitations, we need to add certain elements:

1. Positional encoding	addresses lack of sequence information
2. Multi-headed attention	allows querying multiple positions at each layer
3. Adding nonlinearities	so far, each successive layer is *linear* in the previous one
4. Masked decoding	how to prevent attention lookups into the future?

Motivation for Multi-Head Attention

A single attention function (e.g., scaled dot-product) performs a weighted sum over the value **vectors**, conditioned on the similarity between queries and keys.

However, it is inherently limited in its capacity to model multiple types of dependencies (e.g., syntactic vs. semantic relations, short- vs. long-range dependencies) within a single attention head.

Multi-head attention, addresses a central limitation of single-head attention: the inability to capture **diverse types of dependencies** and interactions within a sequence using a single projection space. It solves this by enabling the model to attend to different representation subspaces simultaneously, enriching the model's ability to learn complex language phenomena.

Intuition

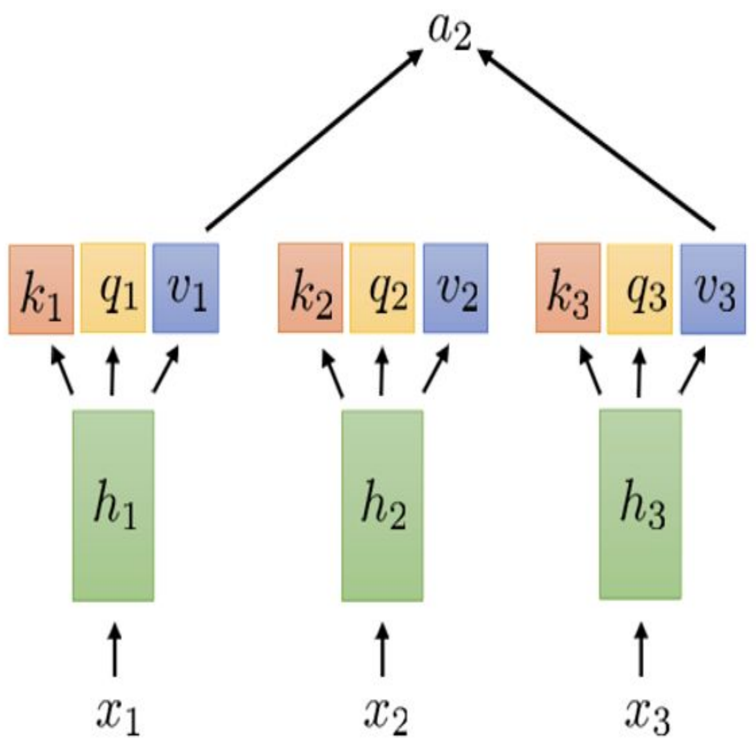
Each head learns to focus on different aspects of the sequence.

- One head might capture **coreference**.
- Another might capture *syntactic structure*.
- Others may capture relative position or *discourse-level information.*

This *parallelism* and *specialization* improves model performance and generalization

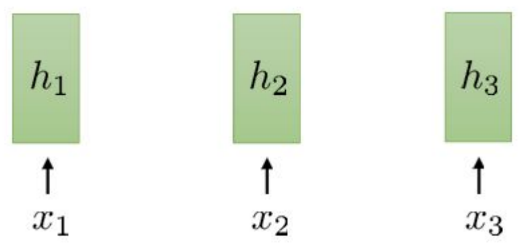
Singe Head Attention

Given that we're fully depending on attention now, it could be beneficial to include more than one time step.



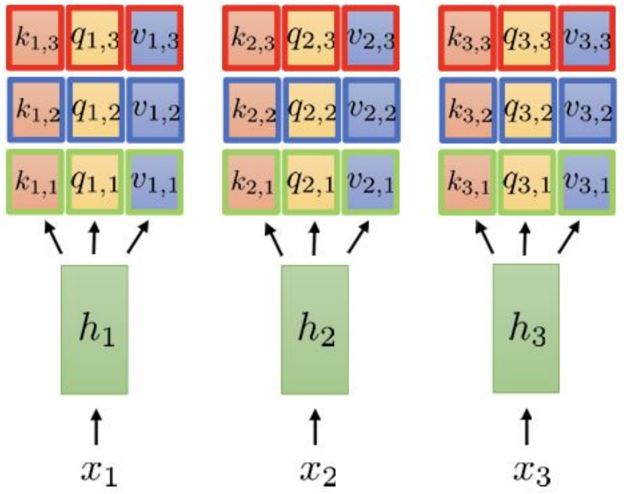
Multi-Head Attention

Solution: Use multiple keys, queries, and values for each time step

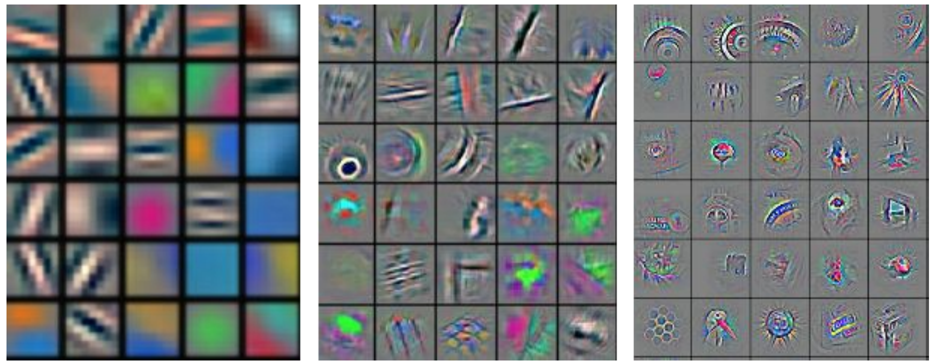


Multi-Head Attention

Solution: Use multiple keys, queries, and values for each time step



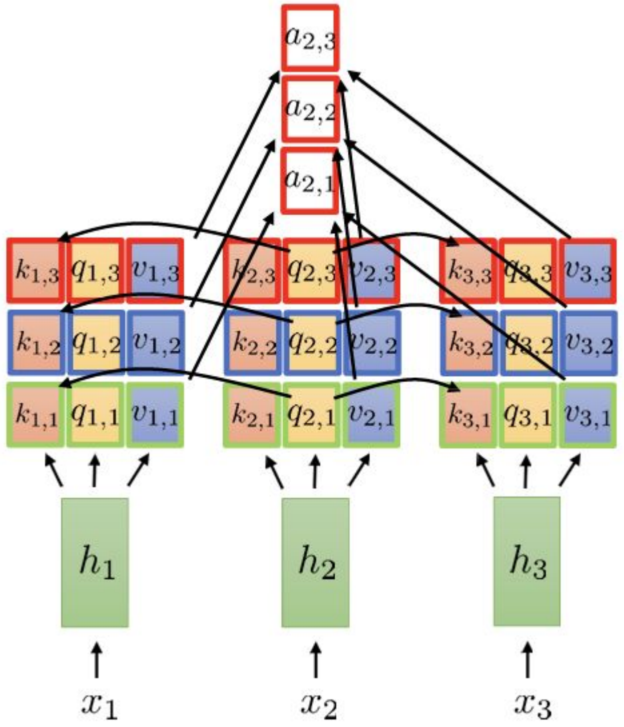
$k_{1,3}$ $q_{1,3}$ $v_{1,3}$
$k_{2,3}$ $q_{2,3}$ $v_{2,3}$
$k_{3,3}$ $q_{3,3}$ $v_{3,3}$
$k_{1,2}$ $q_{1,2}$ $v_{1,2}$
$k_{2,2}$ $q_{2,2}$ $v_{2,2}$
$k_{3,2}$ $q_{3,2}$ $v_{3,2}$
$k_{1,1}$ $q_{1,1}$ $v_{1,1}$
$k_{2,1}$ $q_{2,1}$ $v_{2,1}$
$k_{3,1}$ $q_{3,1}$ $v_{3,1}$
h_1
h_2
h_3
x_1
x_2
x_3



CNN learn at diff. layer

Multi-Head Attention

Solution: Use multiple keys, queries, and values for each time step



$a_{2,3}$
$a_{2,2}$
$a_{2,1}$
$k_{1,3}$ $q_{1,3}$ $v_{1,3}$
$k_{2,3}$ $q_{2,3}$ $v_{2,3}$
$k_{3,3}$ $q_{3,3}$ $v_{3,3}$
$k_{1,2}$ $q_{1,2}$ $v_{1,2}$
$k_{2,2}$ $q_{2,2}$ $v_{2,2}$
$k_{3,2}$ $q_{3,2}$ $v_{3,2}$
$k_{1,1}$ $q_{1,1}$ $v_{1,1}$
$k_{2,1}$ $q_{2,1}$ $v_{2,1}$
$k_{3,1}$ $q_{3,1}$ $v_{3,1}$
h_1
h_2
h_3
x_1
x_2
x_3

full attention vector formed by concatenation:

$$a_2 = \begin{bmatrix} a_{2,1} \\ a_{2,2} \\ a_{2,3} \end{bmatrix}$$

compute weights **independently** for each head

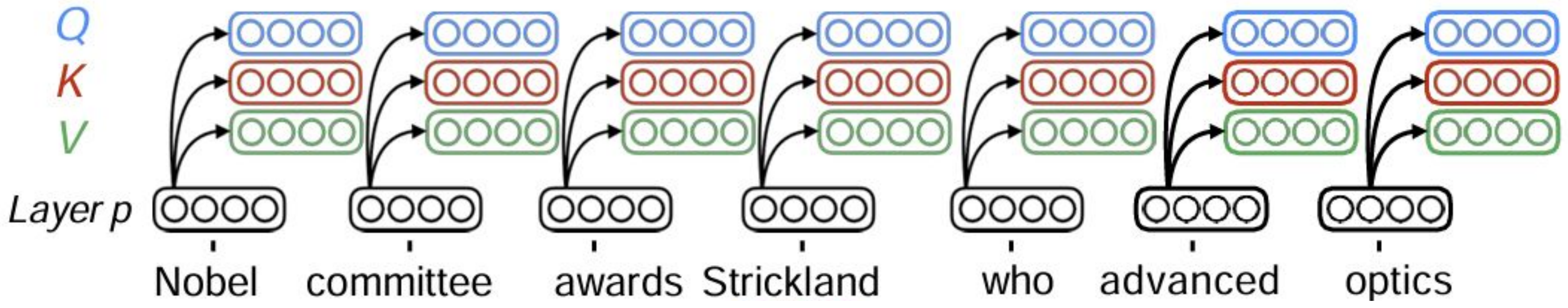
$$e_{l,t,i} = q_{l,i} \cdot k_{l,i}$$

$$\alpha_{l,t,i} = \exp(e_{l,t,i}) / \sum_{t'} \exp(e_{l,t',i})$$

$$a_{l,i} = \sum_t \alpha_{l,t,i} v_{t,i}$$

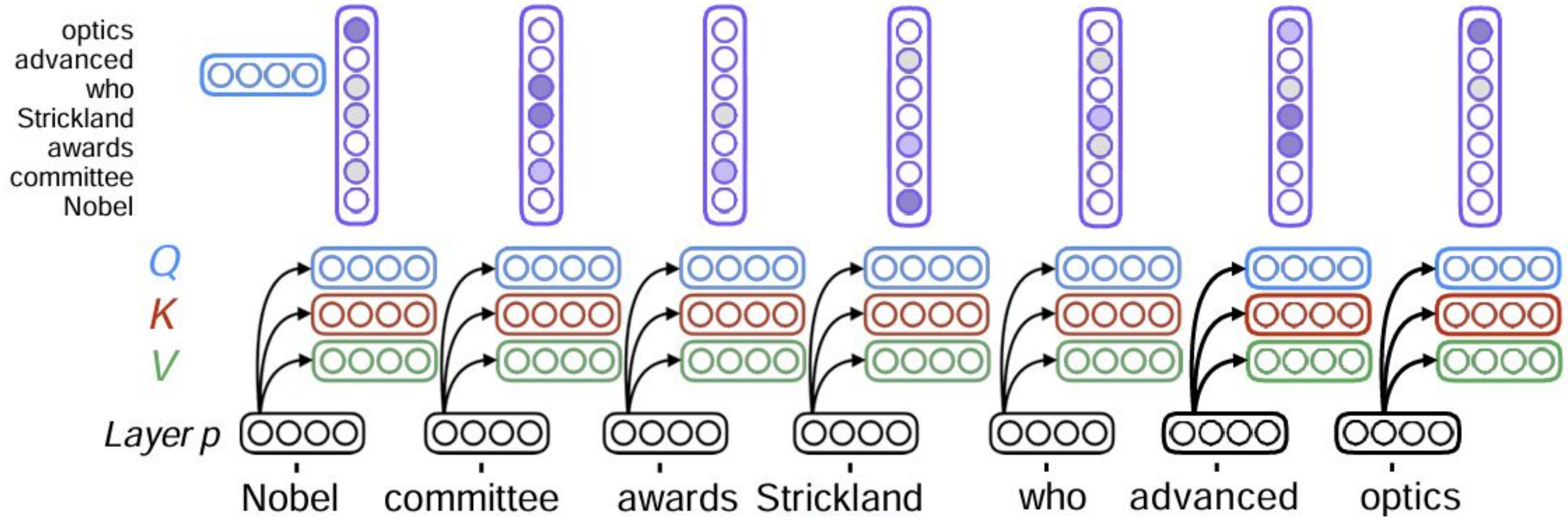
Self-Attention (In Encoder)

Creating Q, K, V



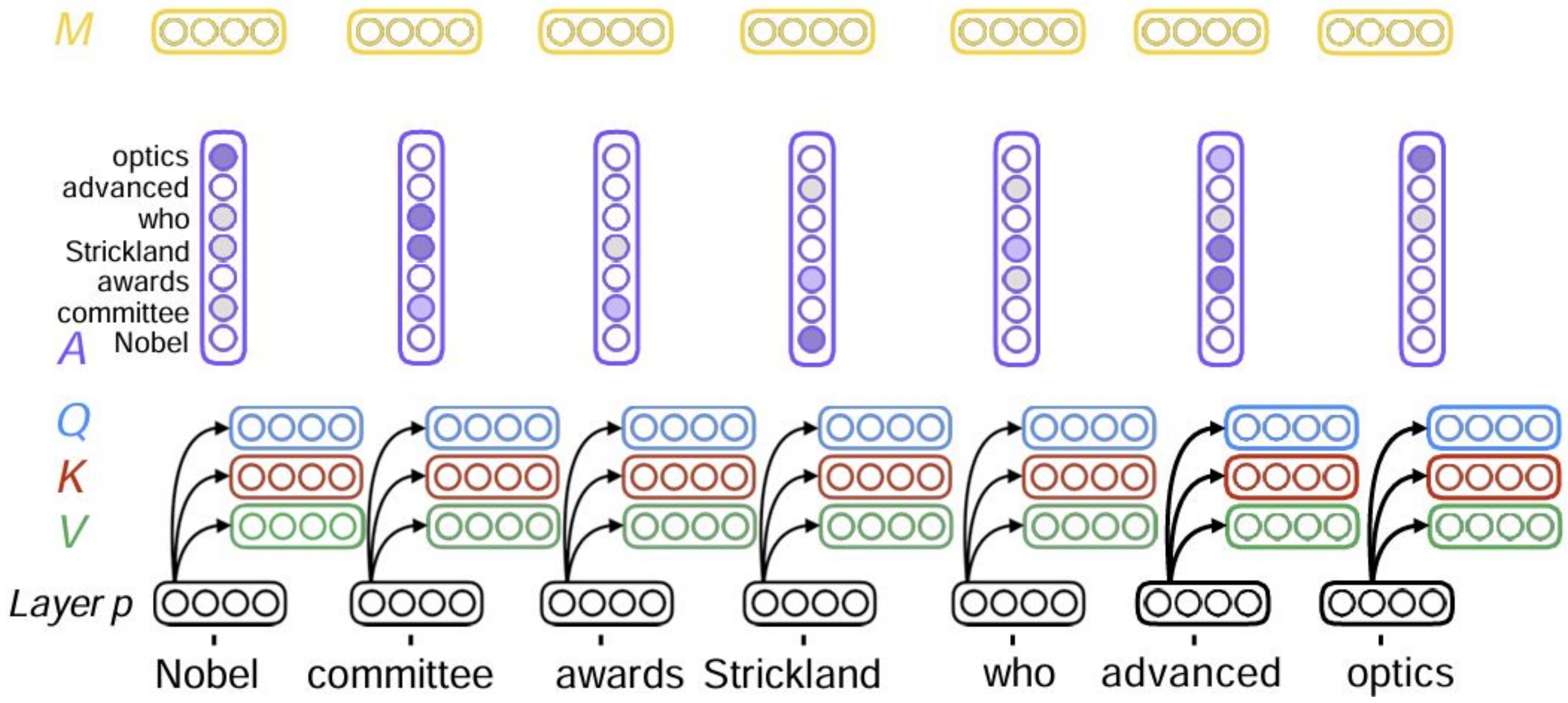
Self-Attention (In Encoder)

Attention score



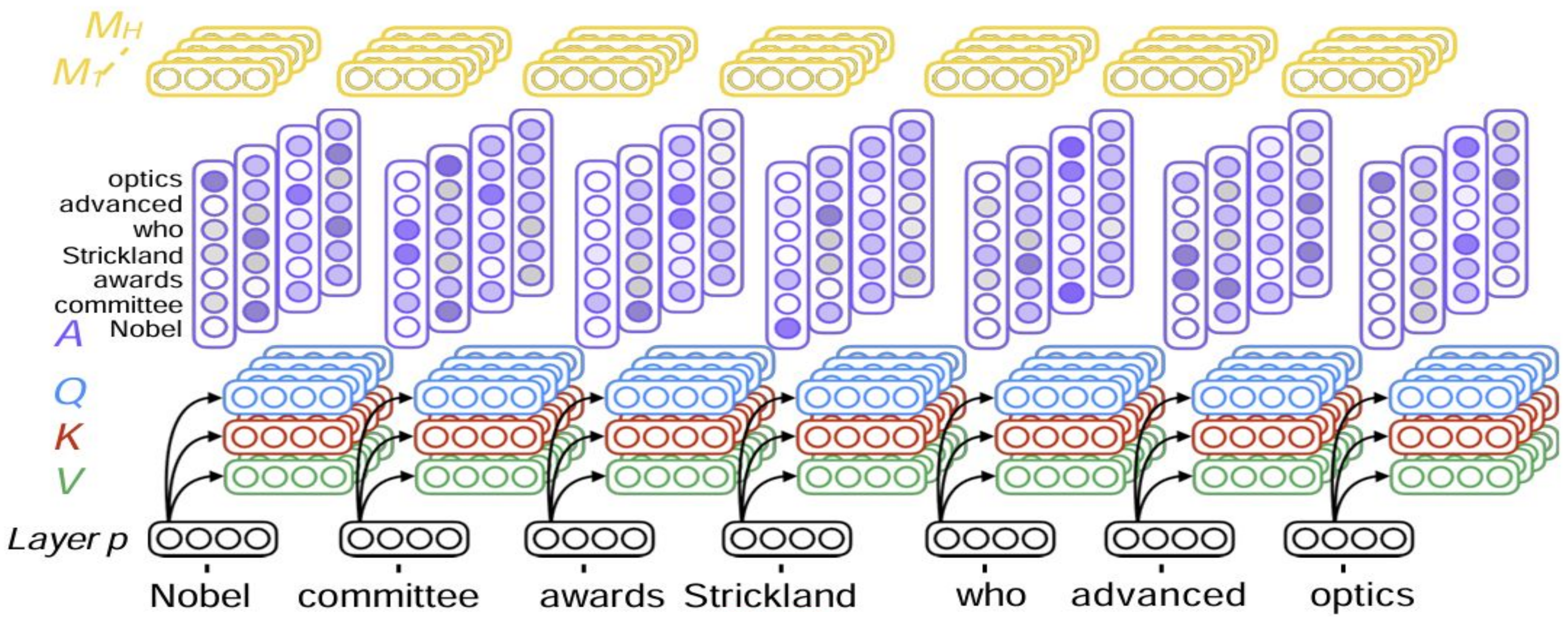
Self-Attention (In Encoder)

Attention vector



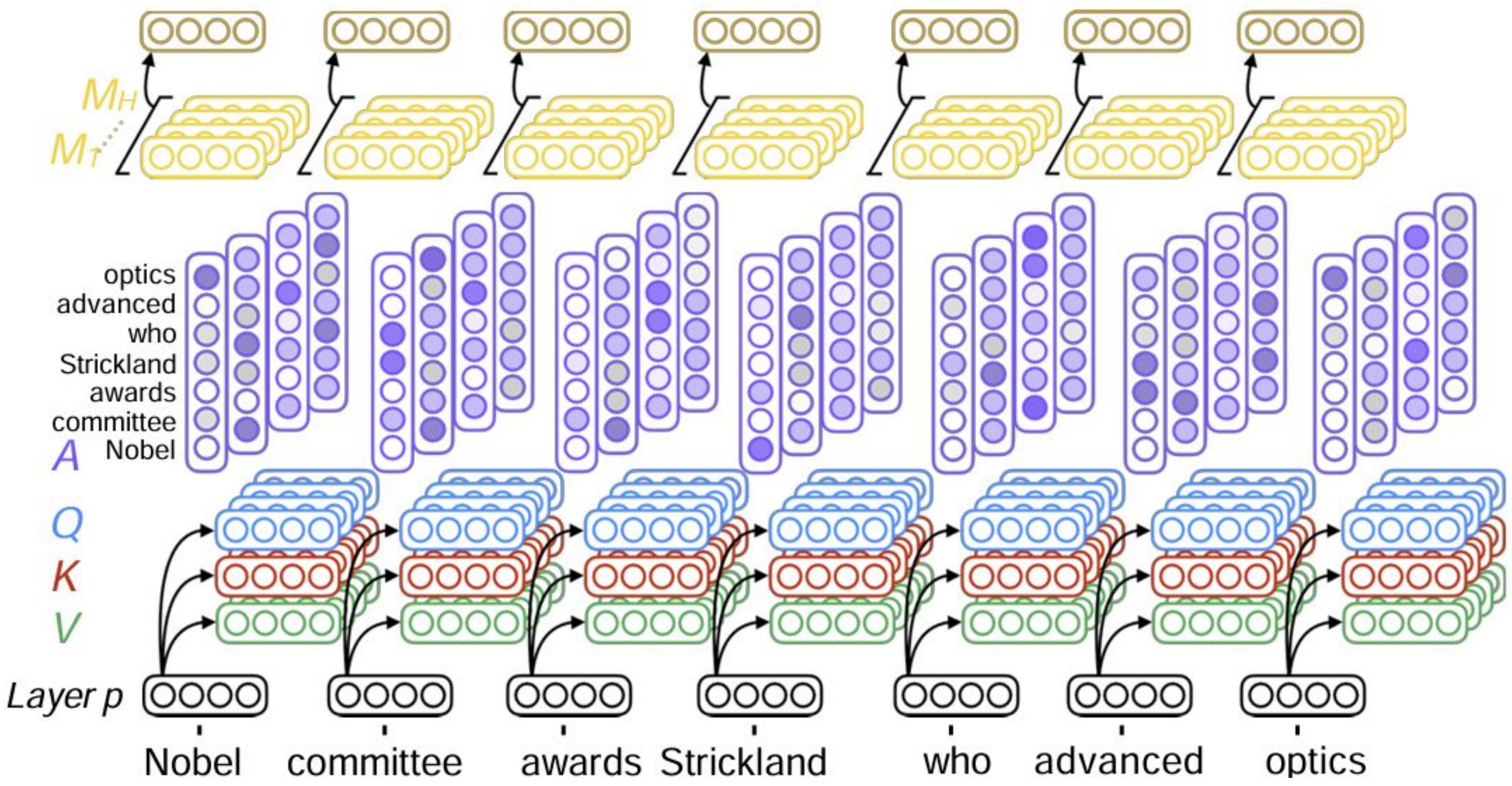
Multi-Head Self-Attention

Multi-head

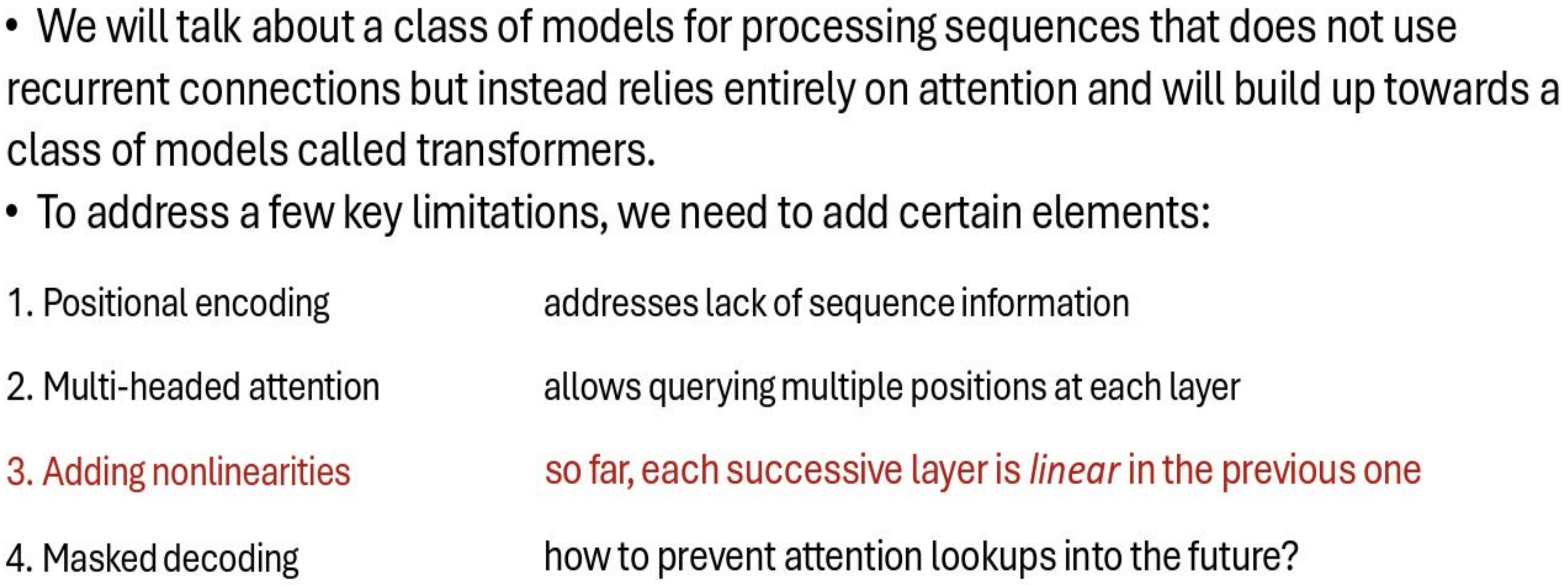


Multi-Head Self-Attention

Concatenate



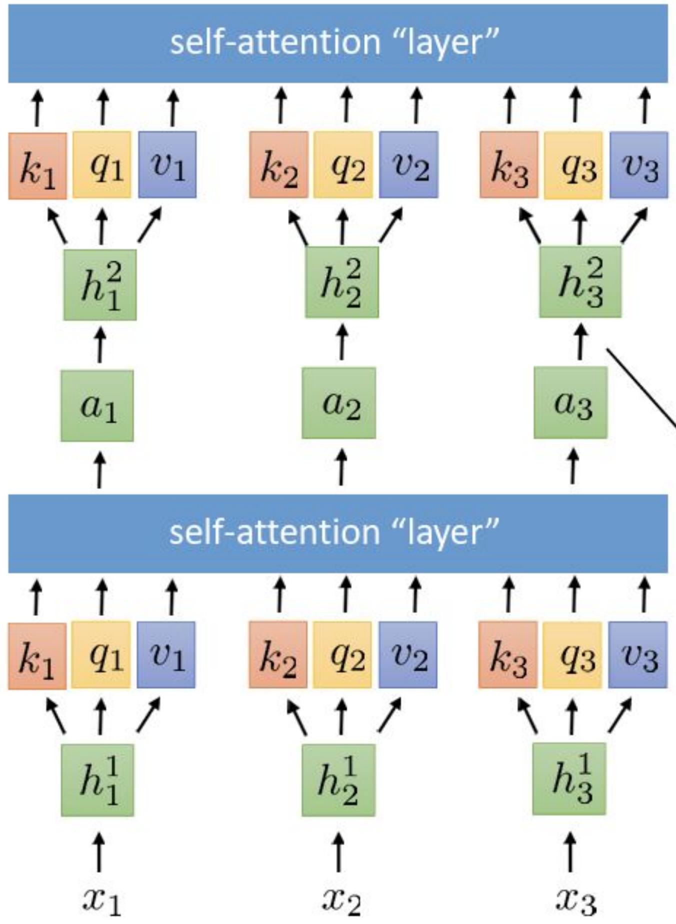
From Self-Attention to Transformers



- We will talk about a class of models for processing sequences that does not use recurrent connections but instead relies entirely on attention and will build up towards a class of models called transformers.
- To address a few key limitations, we need to add certain elements:

1. Positional encoding	addresses lack of sequence information
2. Multi-headed attention	allows querying multiple positions at each layer
3. Adding nonlinearities	so far, each successive layer is *linear* in the previous one
4. Masked decoding	how to prevent attention lookups into the future?

Self-Attention Is "Linear"



The standard self-attention mechanism computes:

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

- Linear in the value vectors $\mathbf{V}$
- Softmax introduces only a shallow non-linearity
- Entire mechanism is a weighted sum — fundamentally linear

Problem: Every self-attention layer is a linear transformation of the previous layer with non-linear weights.

Self-attention layers are largely **linear** in their transformations.

Stacking only linear layers leads to a model with **limited expressivity**.

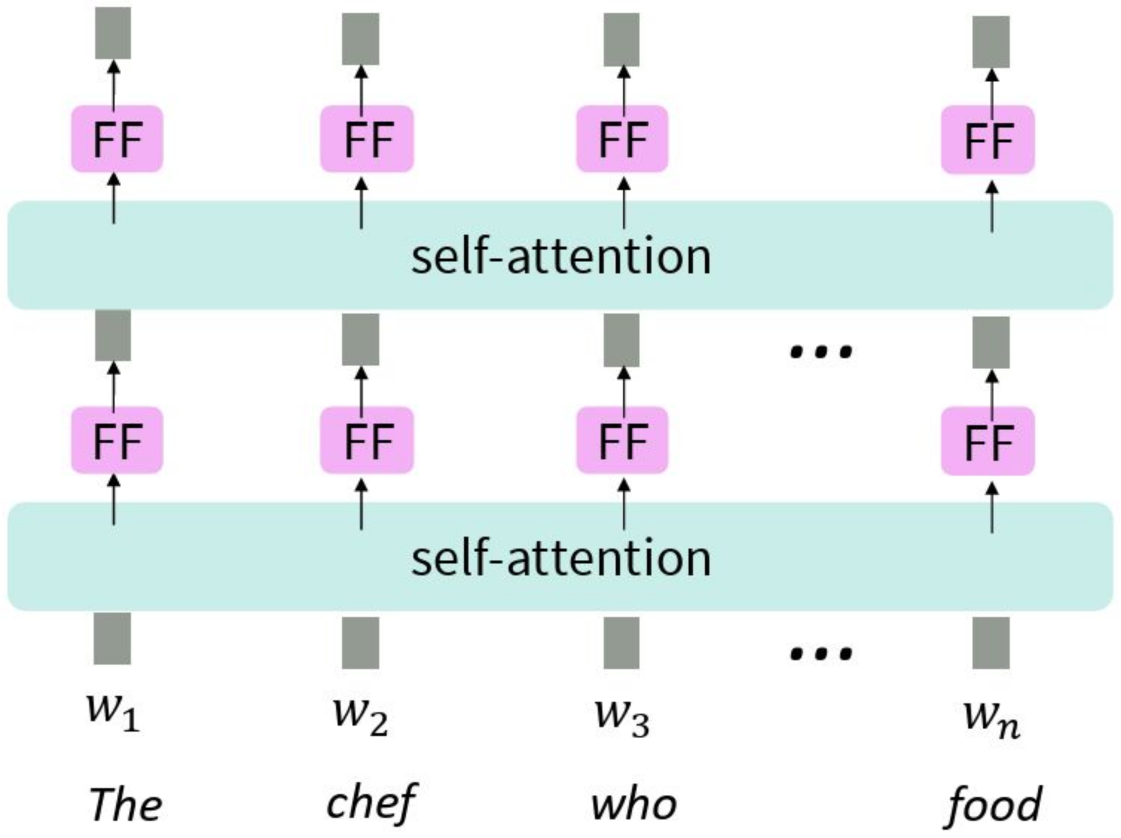
To address this, Transformers alternate attention with non-linear components.

Position-wise Feed-Forward Networks

- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages **value** vectors

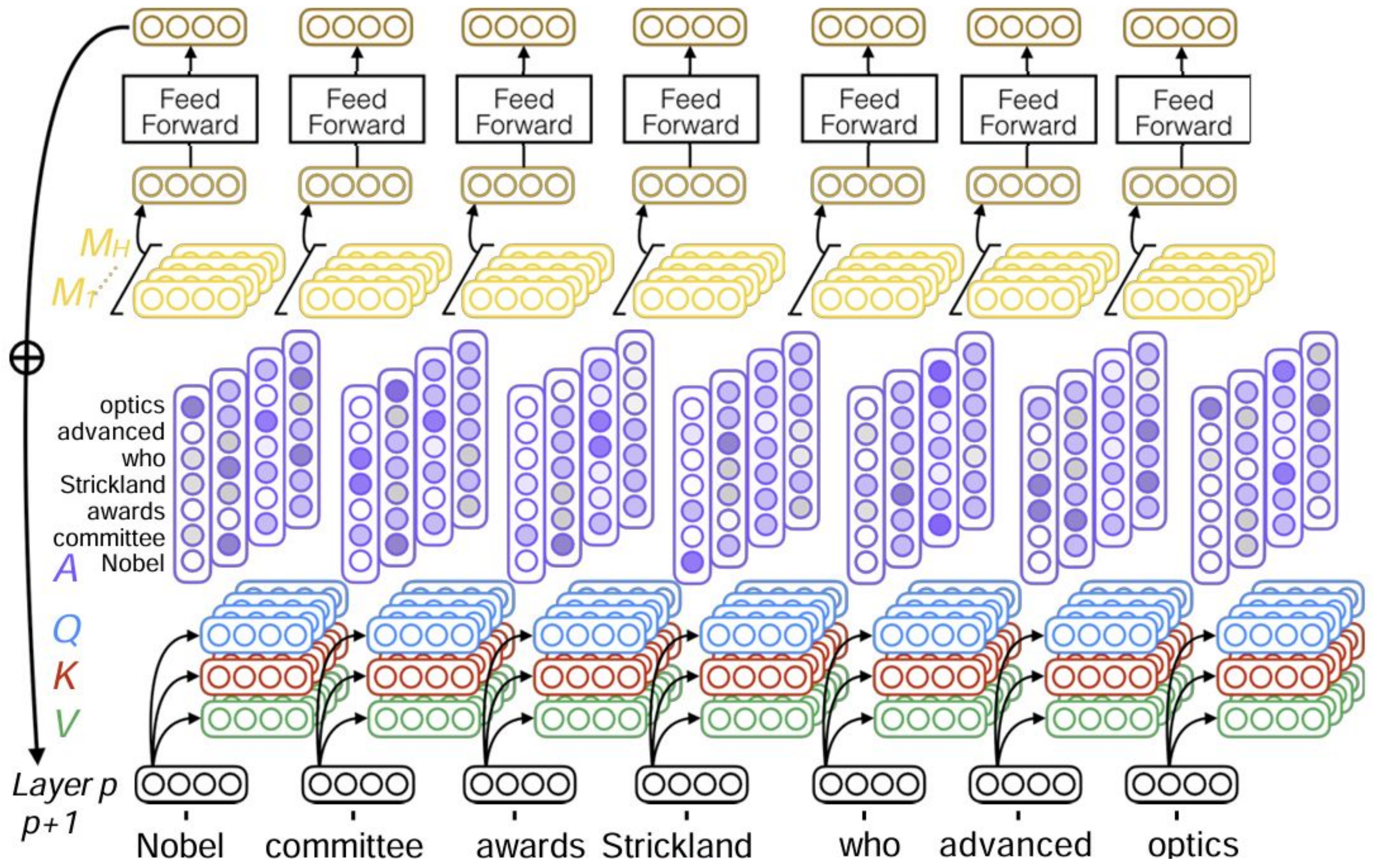
- Easy fix: add a **feed-forward network** to post-process each output vector.

$$
\begin{aligned}
m_i &= MLP(\text{output}_i) \\
&= W_2 * \text{ReLU}(W_1\ \text{output}_i + b_1) + b_2
\end{aligned}
$$

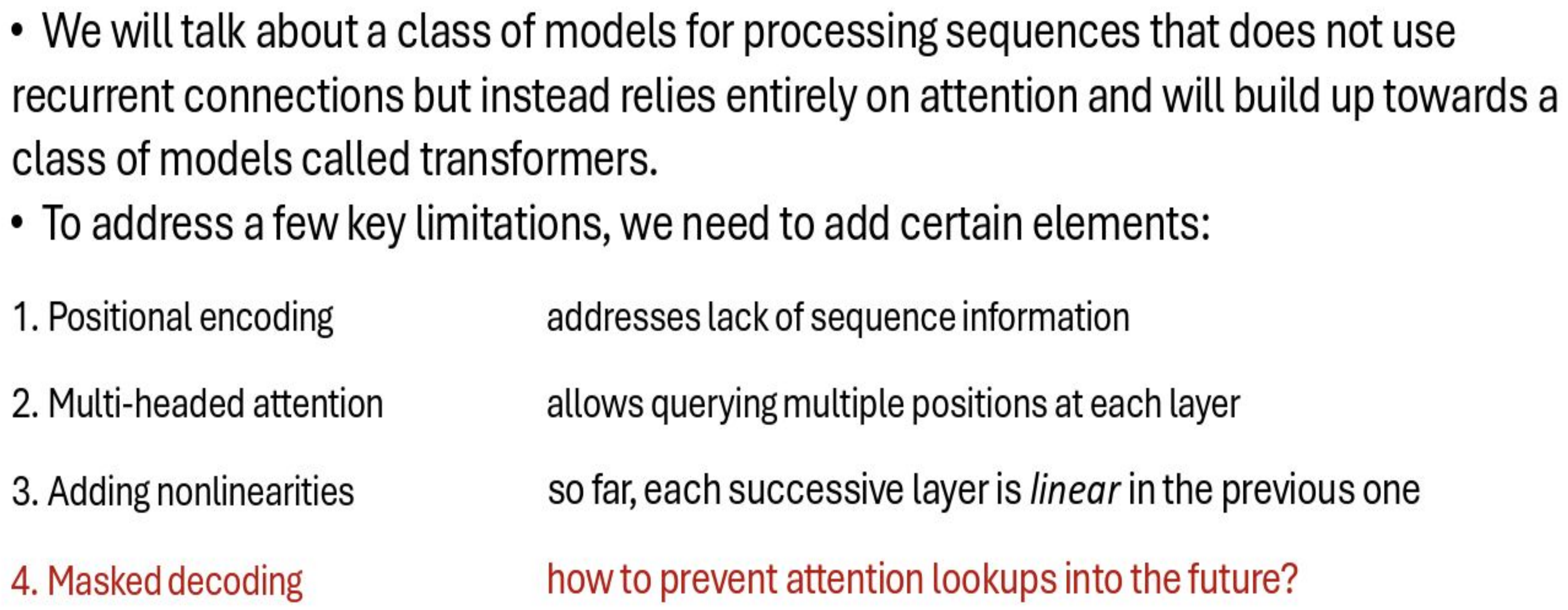


Intuition: the FF network processes the result of attention

Position-wise Feed-Forward Networks



From Self-Attention to Transformers



- We will talk about a class of models for processing sequences that does not use recurrent connections but instead relies entirely on attention and will build up towards a class of models called transformers.
- To address a few key limitations, we need to add certain elements:

1. Positional encoding	addresses lack of sequence information
2. Multi-headed attention	allows querying multiple positions at each layer
3. Adding nonlinearities	so far, each successive layer is *linear* in the previous one
4. Masked decoding	how to prevent attention lookups into the future?

Barriers and solutions for Self-Attention as a building block

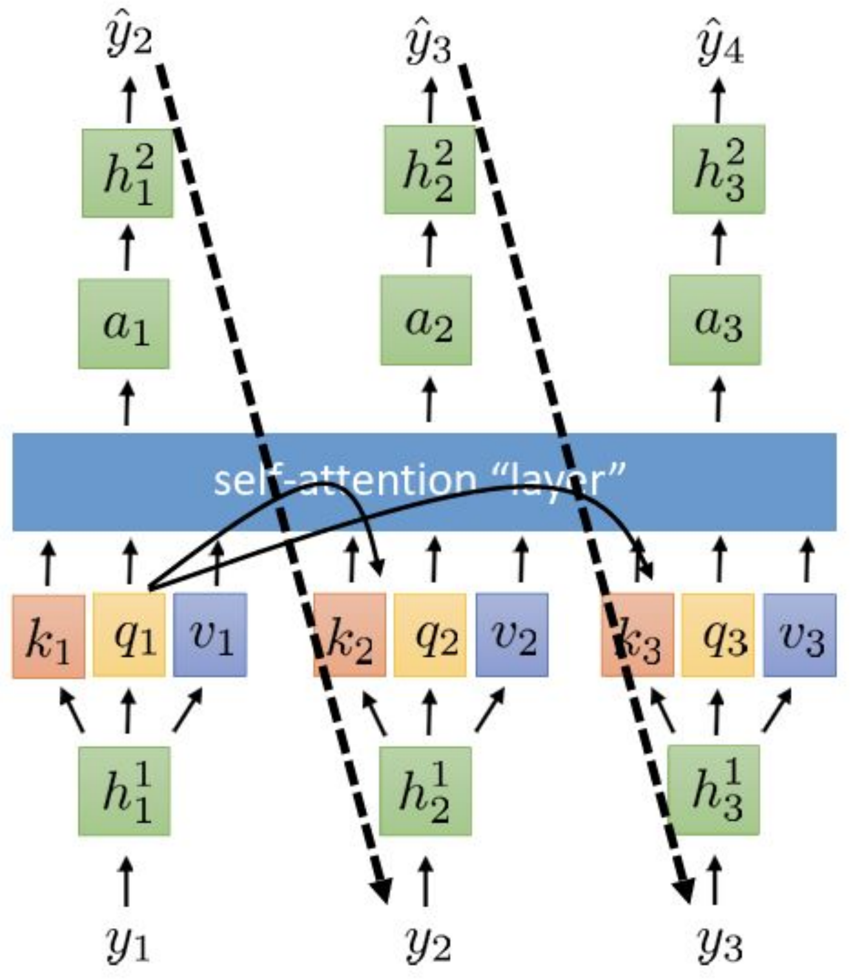
Barriers

- Doesn't have an inherent notion of order!
- No nonlinearities for deep learning magic! It's all just weighted averages
- Need to ensure we don't "look at the future" when predicting a sequence
  - Like in machine translation
  - Or language modeling

Solutions

- Add position representations to the inputs
- Easy fix: apply the same feedforward network to each self-attention output.

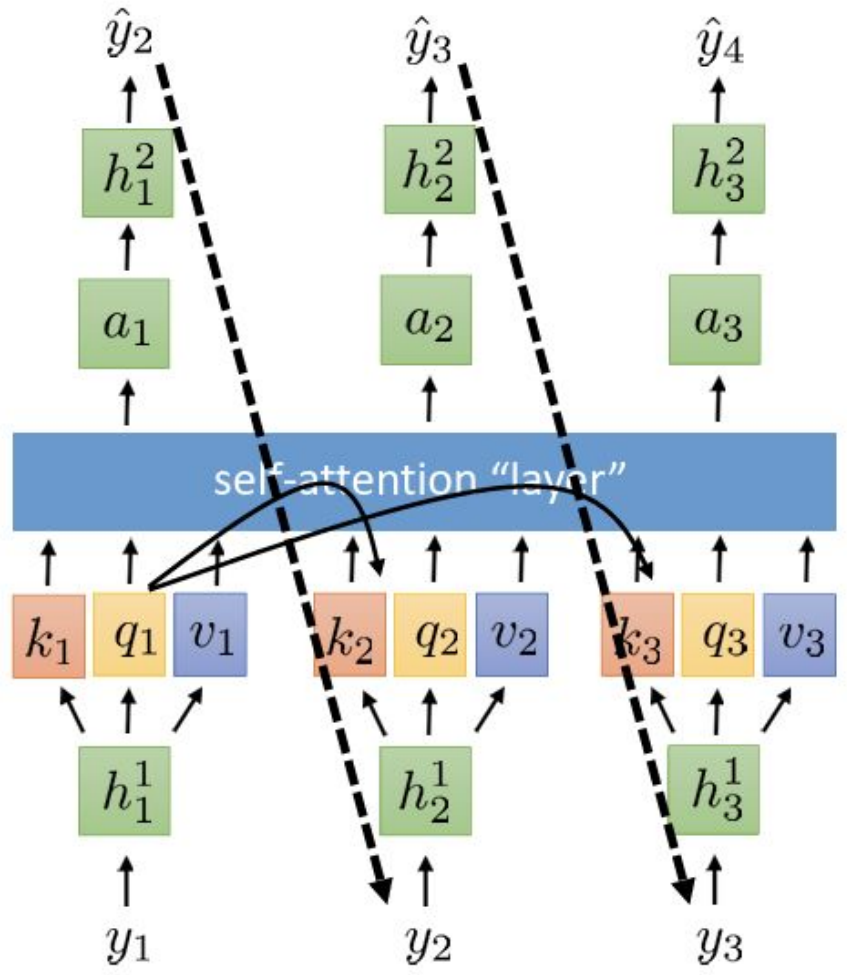
Self-attention can see the future!



A **crude** self-attention "language model":

In practice, there would be several alternating self-attention layers and position-wise feedforward networks

Self-attention can see the future!



A **crude** self-attention "language model":

In practice, there would be several alternating self-attention layers and position-wise feedforward networks

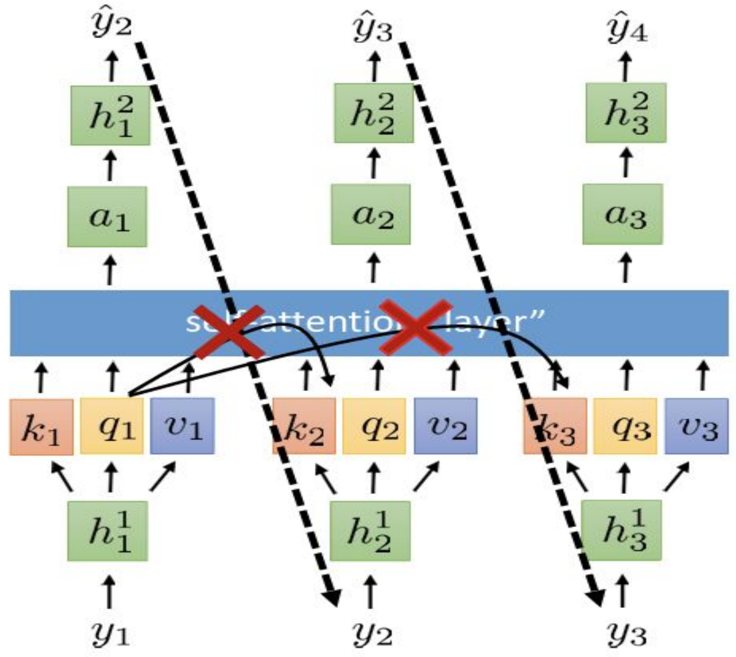
Big problem: self-attention at step 1 can look at the value at steps 2 & 3, which is based on the **inputs** at steps 2 & 3

At test time (when decoding), the **inputs** at steps 2 & 3 will be based on the output at step 1…

…which requires knowing the **input** at steps 2 & 3

Masked Attention

A **crude** self-attention "language model":



At test time (when decoding), the **inputs** at steps 2 & 3 will be based on the output at step 1...

...which requires knowing the **input** at steps 2 & 3

Must allow self-attention into the **past**...

...but not into the **future**

What is Masked Attention?

In standard self-attention, each token can attend to **all** other tokens.

In language generation, we generate tokens **one at a time**.

Masked attention prevents a token from seeing **future tokens**.

Key idea: Each position can only attend to itself and tokens before it.

Why Do We Need Masked Attention?

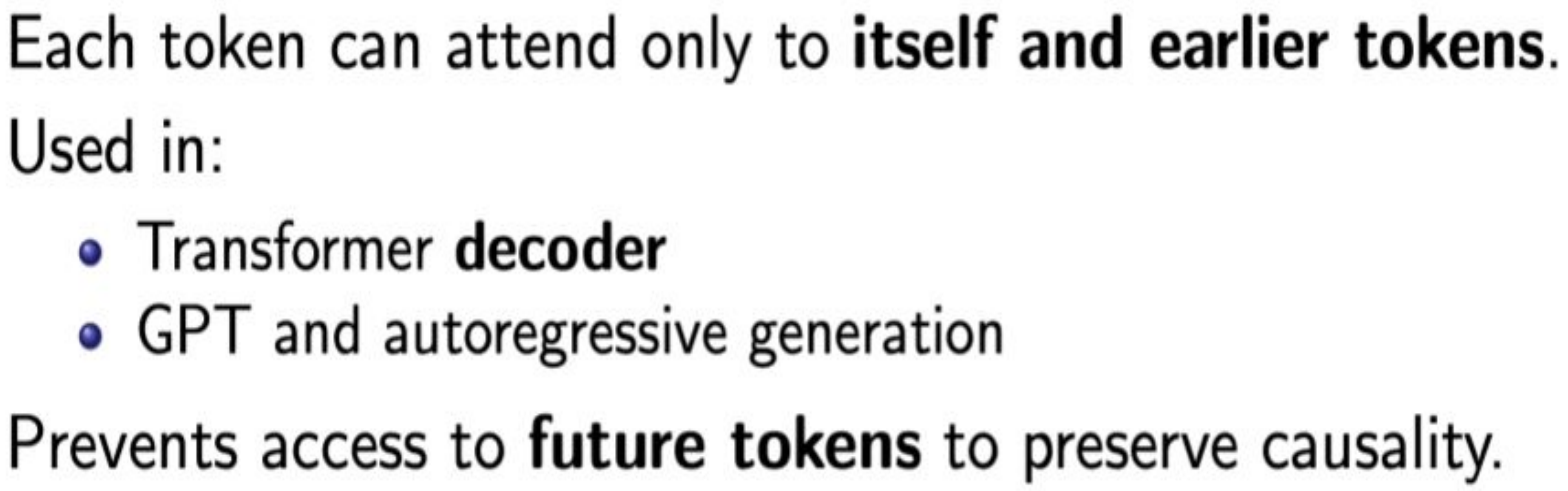
To avoid **cheating** during training: the model shouldn't use future words to predict the current one.

To preserve **causality** in autoregressive models like GPT.

Ensures the model mimics real text generation — **left-to-right**.

Without masking: token y_1 might access y_2, y_3 — which are unknown at generation time.

Masked Attention Summary



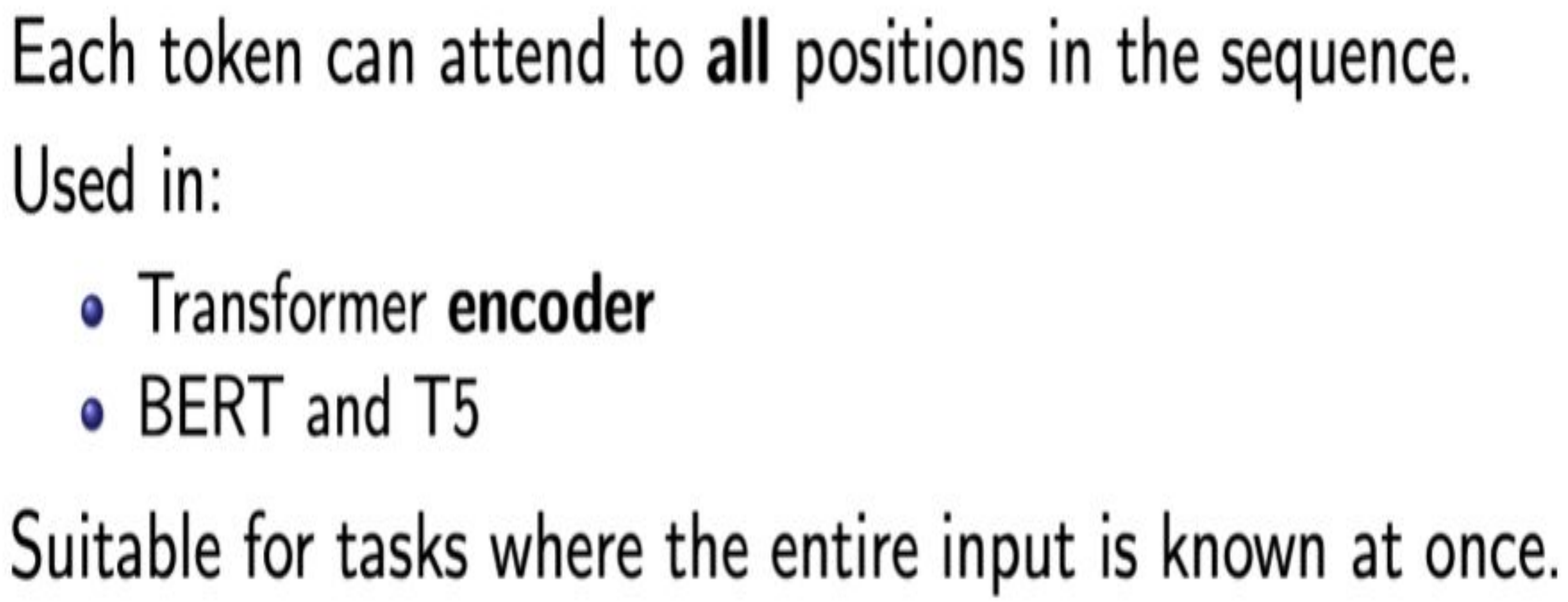
Each token can attend only to **itself and earlier tokens**.

Used in:

- Transformer **decoder**
- GPT and autoregressive generation

Prevents access to **future tokens** to preserve causality.

Unmasked Attention



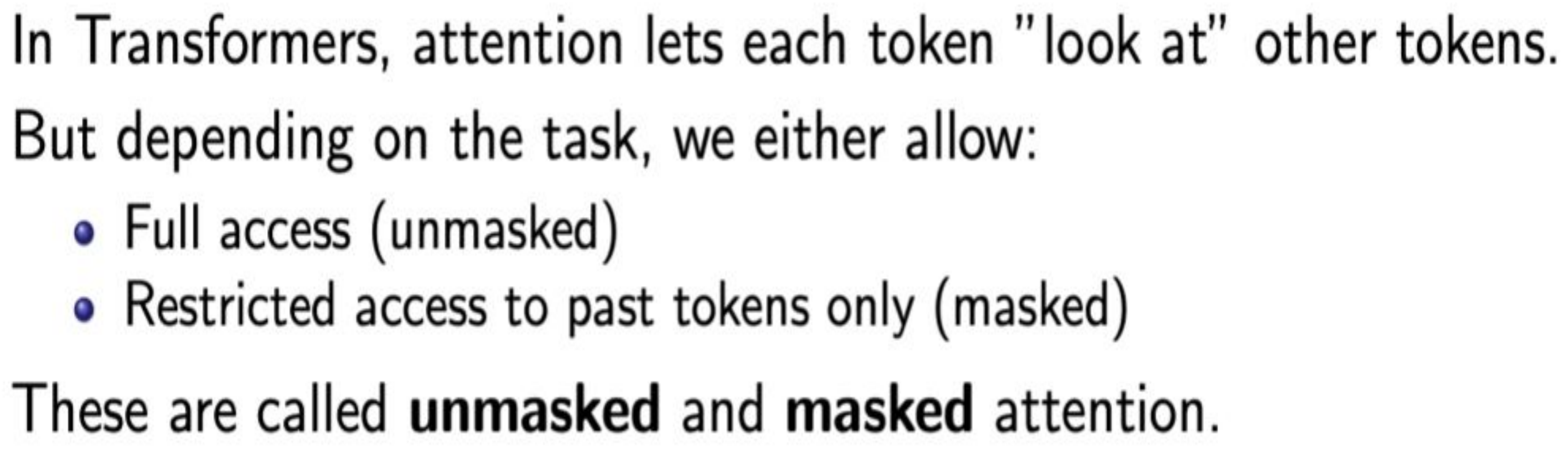
Each token can attend to **all** positions in the sequence.

Used in:

- Transformer **encoder**
- BERT and T5

Suitable for tasks where the entire input is known at once.

Attention Summary



In Transformers, attention lets each token "look at" other tokens.

But depending on the task, we either allow:

- Full access (unmasked)
- Restricted access to past tokens only (masked)

These are called **unmasked** and **masked** attention.

How Decoder attend Encoder?

Cross-Attention

Cross-attention allows the decoder to **attend to the encoder outputs**.

It helps the decoder retrieve relevant information from the source sequence during generation.

Queries come from the decoder; Keys and Values come from the encoder.

$$\text{CrossAttention}(\mathbf{Q}_{\text{dec}}, \mathbf{K}_{\text{enc}}, \mathbf{V}_{\text{enc}}) = \text{softmax}\left(\frac{\mathbf{Q}_{\text{dec}}\mathbf{K}_{\text{enc}}^{\top}}{\sqrt{d}}\right)\mathbf{V}_{\text{enc}}$$

- $\mathbf{Q}_{\text{dec}}$: decoder query
- $\mathbf{K}_{\text{enc}}, \mathbf{V}_{\text{enc}}$: encoder key/value
- Output: decoder representation influenced by encoder context

Why Cross-Attention Important?

Enables the decoder to **use the encoded input** to generate output.

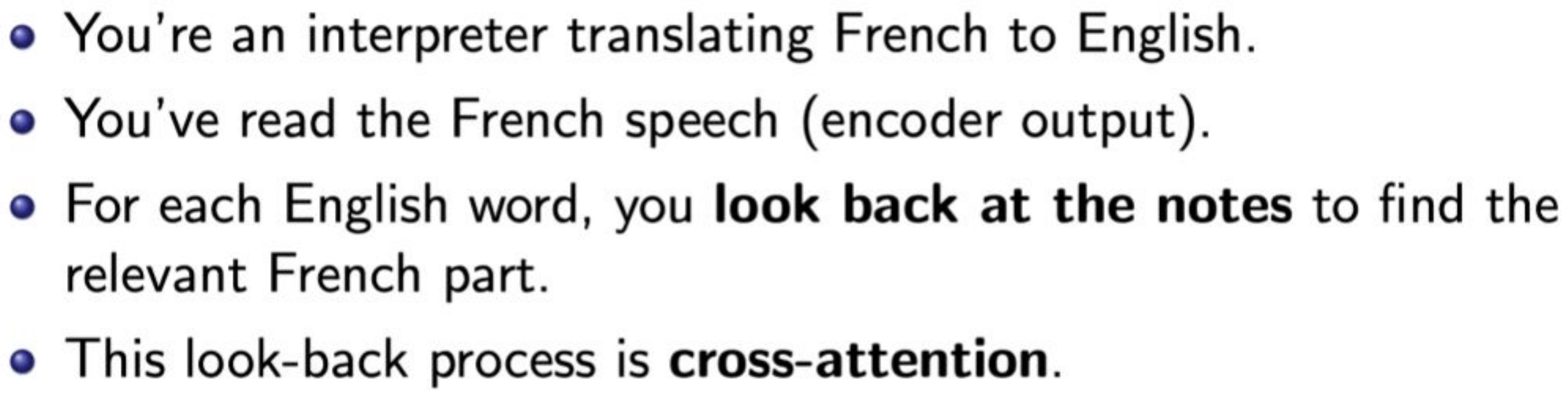
Learns alignments between input and output tokens (e.g., translation).

Supports **conditional generation**: output depends on input.

Example: "Le chat dort" → "The cat sleeps"

- "cat" attends to "chat"
- "sleeps" attends to "dort"

Analogy: Interpreter with Notes

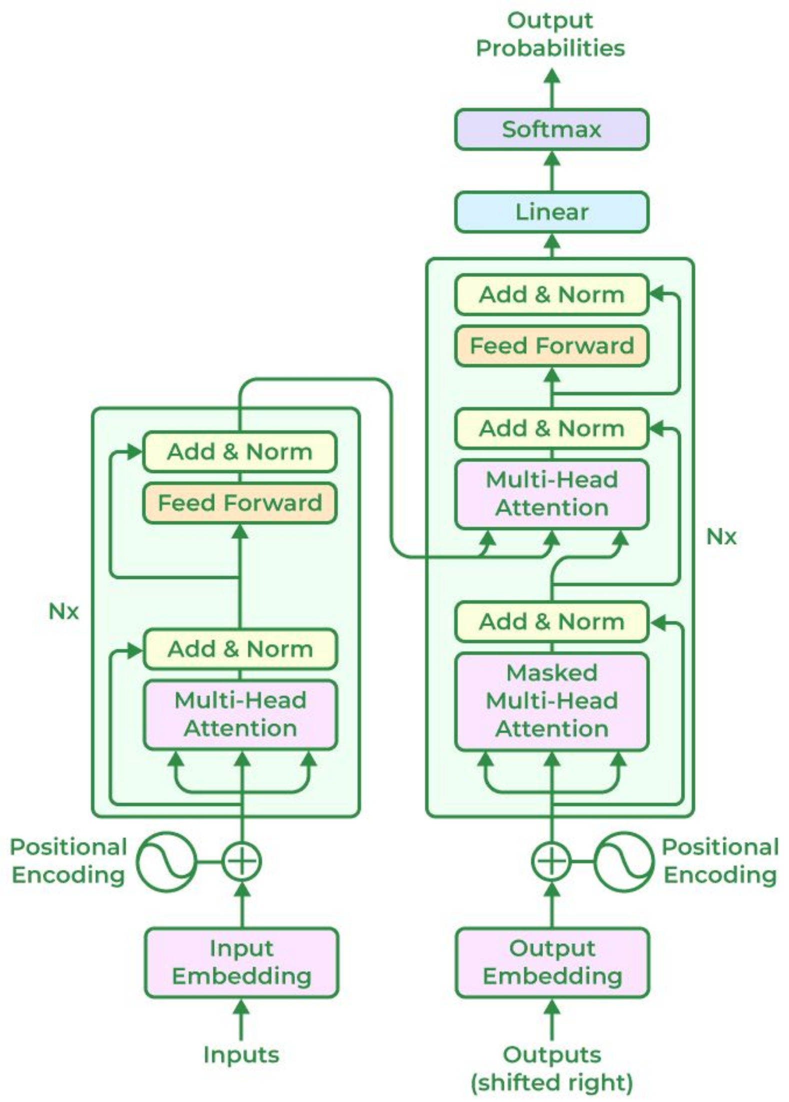


- You're an interpreter translating French to English.
- You've read the French speech (encoder output).
- For each English word, you **look back at the notes** to find the relevant French part.
- This look-back process is **cross-attention**.

Transformer

Inside each decoder layer:

Masked self-attention (left-to-right decoder context)
Cross-attention (decoder attends to encoder output)
Feed-forward network



References

Seq2seq Models With Attention

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

The Illustrated Transformer

https://jalammar.github.io/illustrated-transformer/

IMPERIAL

First half Transformer Lecture

Next

- Position embeddings
- Residual Connections
- Layer Normalization

IMPERIAL

Q and A