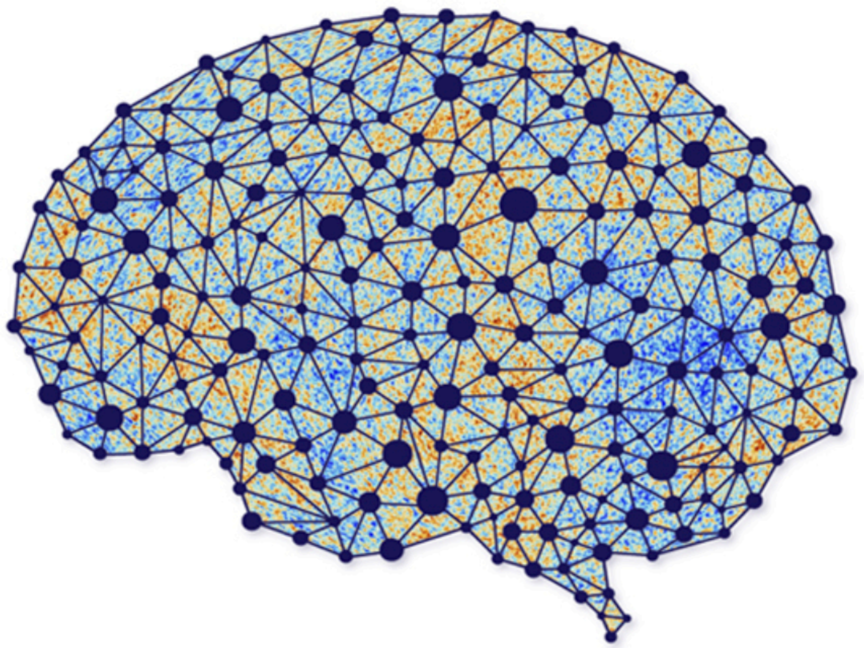
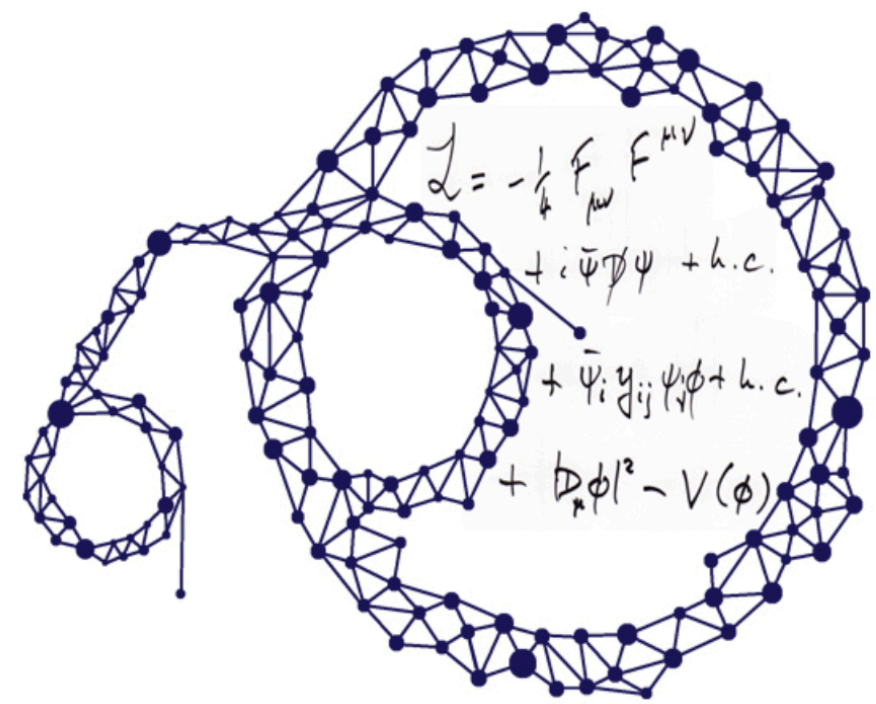
PHY 835: Machine Learning in Physics

Lecture 19: Transformers Part 2

April 4, 2024



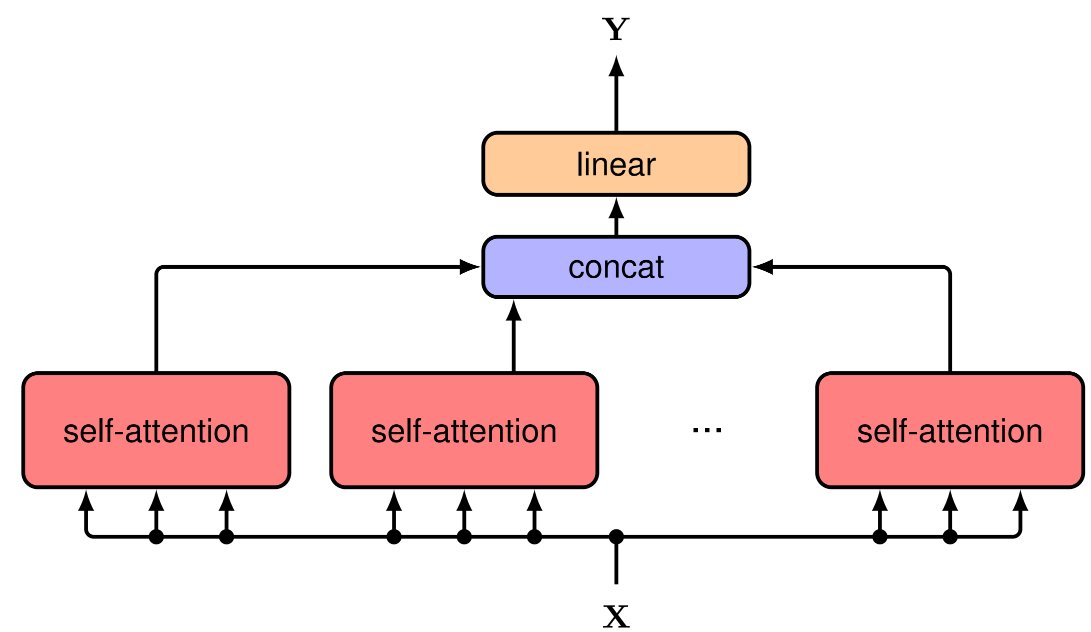
AI
∩
Universe

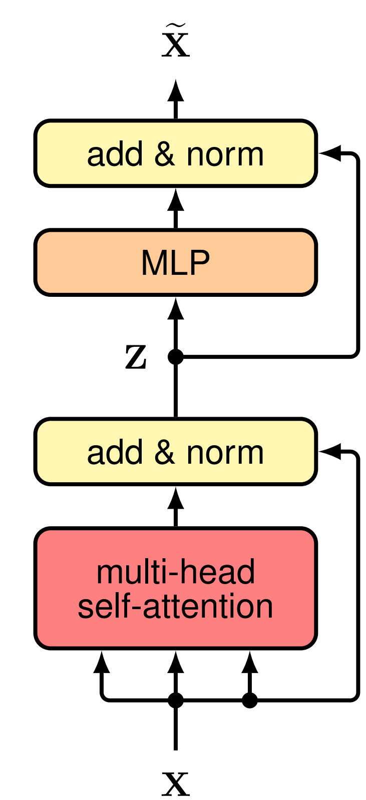


Gary Shiu

Transformer Layers

- NNs benefit greatly from depth, so we can stack self-attention layers (like the right) on top of each other.
- To improve efficiency, transformer layers are followed by **layer normalization**: https://arxiv.org/abs/1607.06450
- Output of an attention layer are constrained to be linear combinations of the inputs, though non-linearities enter through the attention weights.
- Enhance flexibility by post-processing the output of each layer using non-linear network denoted by MLP (e.g., fully connected NN with ReLu activation.





Position Encoding

- The weight matrices $\mathbf{W}^{(q)}, \mathbf{W}^{(k)}, \mathbf{W}^{(v)}$ are shared among the input tokens, so transformer is equivariant w.r.t. input permutations.
- Token ordering is important in sequential processing: *"The professor failed the students"* is different from *"The students failed the professor"*.
- Construct a position encoding vector $\mathbf{r}_n$ and combine with the input token embedding $\mathbf{x}_n$. Concatenation would increase the dim of input space and significantly increase computational cost. Instead:

$$\widetilde{\mathbf{x}}_n = \mathbf{x}_n + \mathbf{r}_n.$$

- The position & input vectors have the same dim. Two randomly chosen uncorrelated vectors tend to be nearly orthogonal in high dim.
- Associate an integer $1,2,3,\ldots$ to each position has the problem of corrupting the input vector because the length is unbounded, and vary among training sets. May not recognize new longer input sequence.

Position Encoding

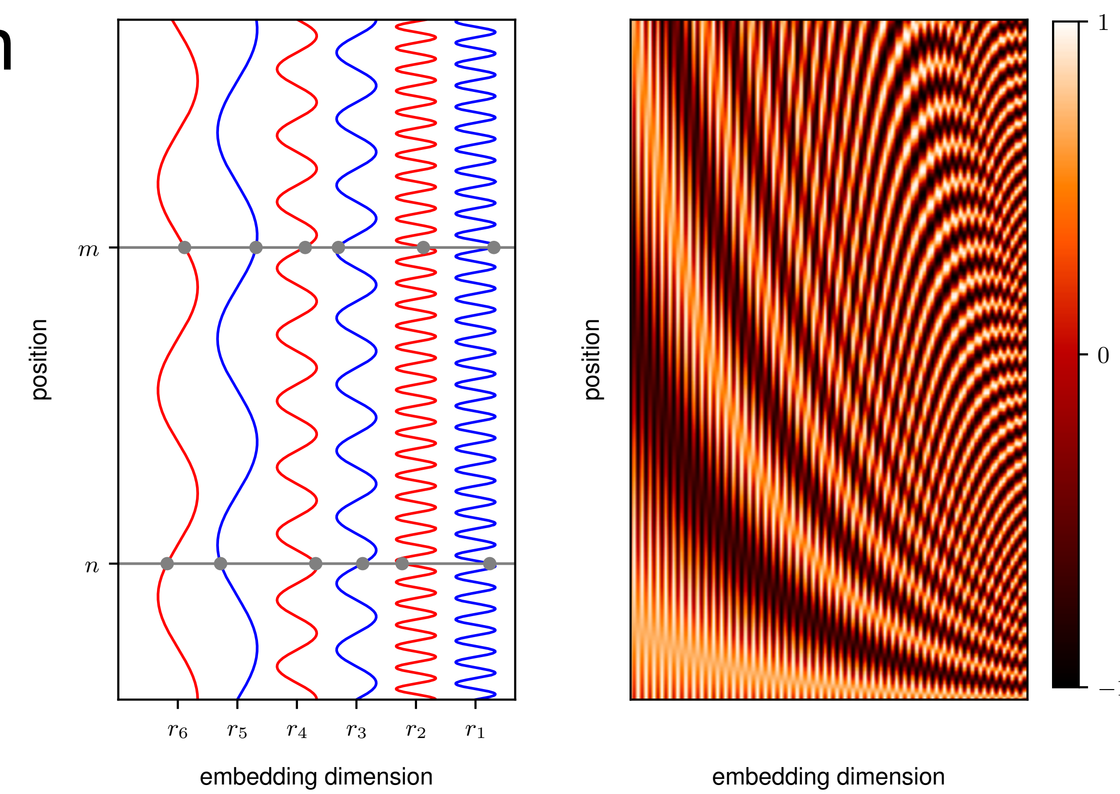
- Assigning a # between $(0,1)$ to each token in the sequence does not work as the rep. is not unique for a given position (depends on sequence length).

- Is there an encoding that provides a unique rep. for each position, is bounded, generalizable to longer sequences, & capture relative positions?

- Use sinusoidal functions (Vaswani et al):

$$r_{ni} = \begin{cases} \sin\left(\dfrac{n}{L^{i/D}}\right), & \text{if } i \text{ is even,} \\ \cos\left(\dfrac{n}{L^{(i-1)/D}}\right), & \text{if } i \text{ is odd.} \end{cases}$$

- Because of the properties of sine and cosine, the encoding allows the network to attend to relative positions.



(a) (b)

Similar to binary reps of integers, except that r_{ni} is continuous:

1 :	0	0	0	1
2 :	0	0	1	0
3 :	0	0	1	1
4 :	0	1	0	0
5 :	0	1	0	1
6 :	0	1	1	0
7 :	0	1	1	1
8 :	1	0	0	0
9 :	1	0	0	1

Transformer for NLP

- A typical NLP pipeline starts with a **tokenizer** that splits the text into words or word fragments. Using words as tokens may not be ideal:
  - Some words (e.g. names, technical terms) aren't in the vocabulary.
  - How about punctuation? A question mark contains info to encode.
  - The vocabulary would need different tokens for different versions of the same word with different suffices (e.g., walk, walks, walked, walking), and there is no way to clarify these variations are related.
- Then each of the tokens is mapped to a learned **embedding**.
  - The whole vocabulary is stored in a matrix $\Omega_e \in \mathbb{R}^{D \times |\mathscr{V}|}$ where $|\mathscr{V}|$ is the vocabulary size; this vocabulary matrix is learned.
- These embeddings are passed thru a series of **transformer layers**.

Tokenization

- One approach is to use letters and punctuations as the vocabulary. But this requires the subsequent network to re-learn the relations between the very small pieces.
- A compromise is **sub-word tokenizer** such as **byte pair encoding** that greedily merges sub-strings based on their frequencies.
- Consider the following nursery rhyme:

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

- The tokens are initially just the characters & whitespace (represented by an underscore), and their frequencies given in the table.

Byte pair encoding

- At each iteration, the sub-word tokenizer looks for the most commonly occurring adjacent pair of tokens and merges them. This creates a new token & decreases the counts for the original tokens.

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

- At the second iteration, the algorithm merges e and the whitespace character_. The last character of the first token to be merged cannot be whitespace, which prevents merging across words.

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	se	a	e_	t	o	h	l	u	b	d	e	w	c	s	f	i	m	n	p	r
21	13	12	12	11	8	6	6	4	3	3	3	3	2	2	1	1	1	1	1	1

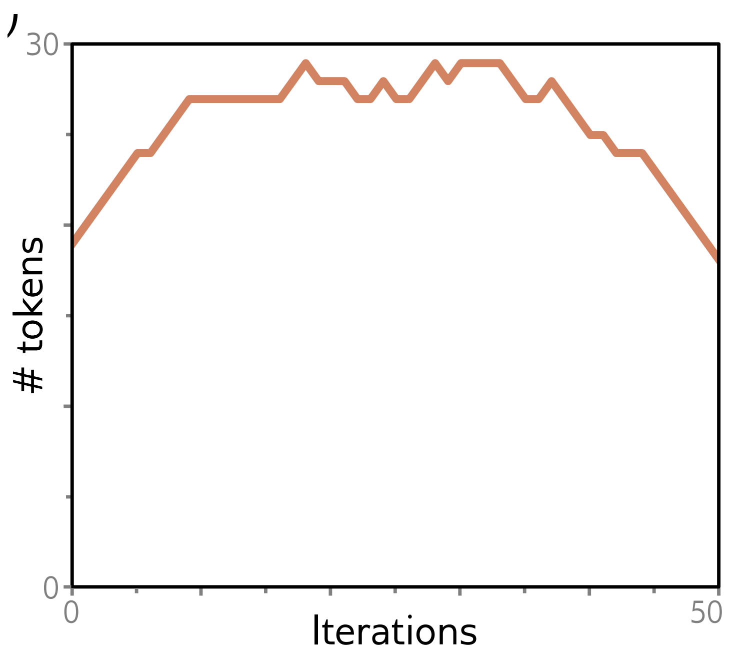
Byte pair encoding (continued)

- After 22 iterations, the tokens consist of a mix of letters, word fragments, and commonly occurring words:

see_	sea_	e	b	l	w	a	could_	hat_	he_	o	t	t_	the_	to_	u	a_	d	f	m	n	p	s	sailor_	to
7	6	4	3	3	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1

- If we continue this process indefinitely, the tokens eventually represent the full words:

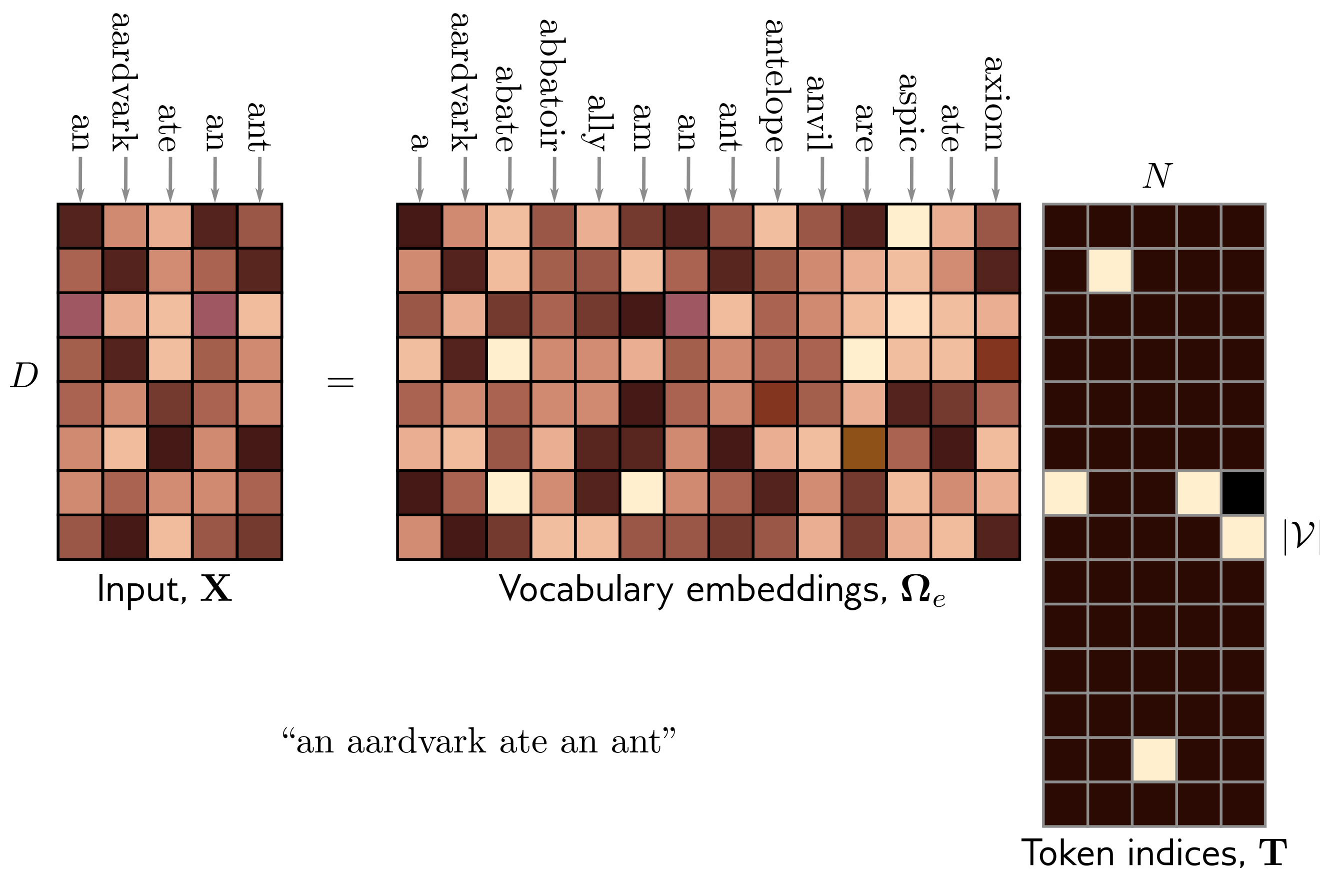
see_	sea_	could_	he_	the_	a_	all_	blue_	bottom_	but_	deep_	of_	sailor_	that_	to_	was_	went_	what_
7	6	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1



The number of tokens increases as we add word fragments to the letters and then decreases again as we merge these fragments.

Embeddings

- Each token is mapped to a unique word embedding; the embeddings for the whole vocabulary are storied in a matrix $\Omega_e \in \mathbb{R}^{D\times|\mathscr{V}|}$



$$\mathbf{X} = \boldsymbol{\Omega}_e \mathbf{T},$$

- The matrix Ω_e is learned like any other network parameter.

- A typical embedding size D is 1024 and a typical total vocabulary size $|\mathscr{V}|$ is $30{,}000$. Many parameters in Ω_e to learn.

Transformer model

- The embedding matrix $\mathbf{X}$ representing the text is passed through a series of K transformer layers, called a **transformer model**.
- Three types of transformer models:
  - An **encoder** transforms the text embeddings into a representation that can support a variety of tasks (e.g., sentiment analysis).
  - A **decoder** predicts the next token to continue the input text.
  - **Encoder-decoder** used in sequence-to-sequence tasks, where one text string is converted into another, e.g., machine translation.
- A hands-on tutorial on transformers in pytorch can be found here: https://peterbloem.nl/blog/transformers

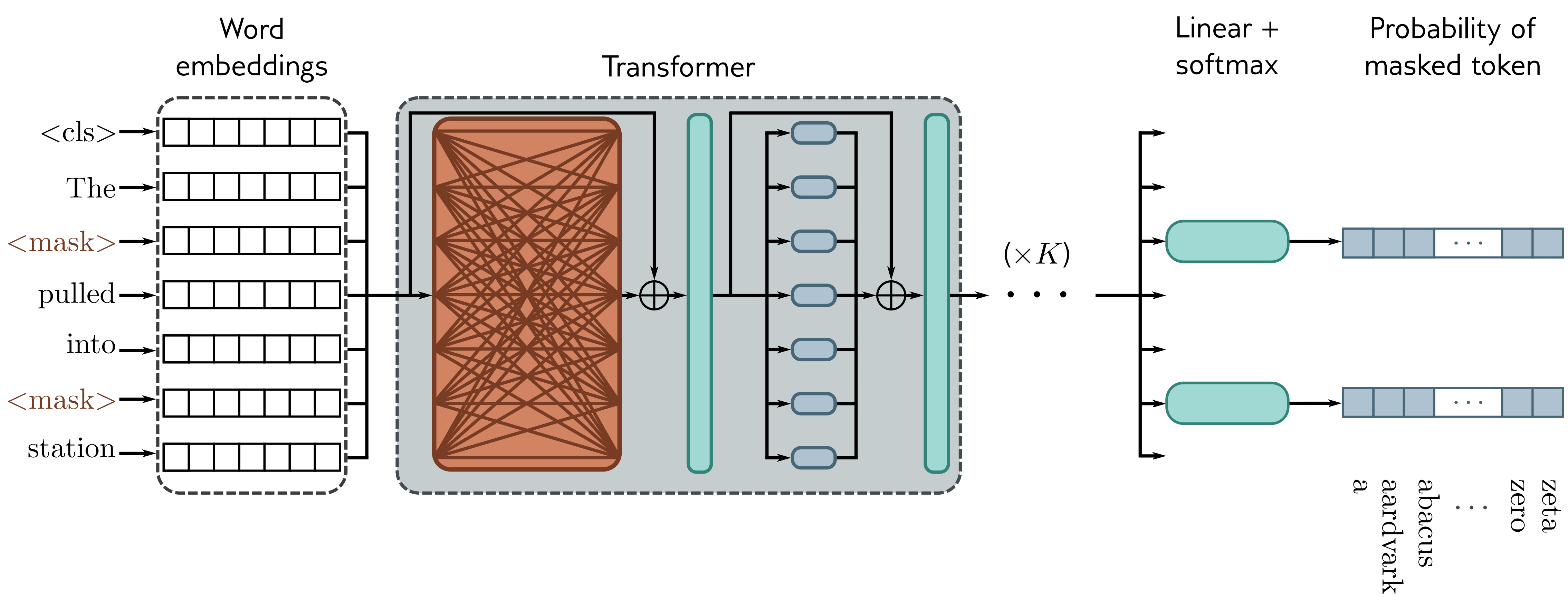
Encoder model example: BERT

https://arxiv.org/abs/1810.04805v2

- BERT is an encoder model that uses a vocabulary of 30,000 tokens.
- Input tokens are converted to 1024 dimensional word embeddings and passed through 24 transformer layers.
- Each contains a self-attention mechanism with 16 heads.
- The weight matrices Q_h, K_h, V_h for each head are 1024×64.
- The total number of parameters is ~ 340 million, but it is now much smaller than state-of-the-art models.
- Encoder models like BERT exploit **transfer learning**: parameters of the ML model are learned during *pre-training* using *self-supervision* from a large corpus of data, followed by a *fine-tuning* stage to adapt for specific task using a smaller body of *supervised training data*.

Pre-training

- For BERT, the self-supervision task consists of predicting missing words from sentences from a large internet corpus.

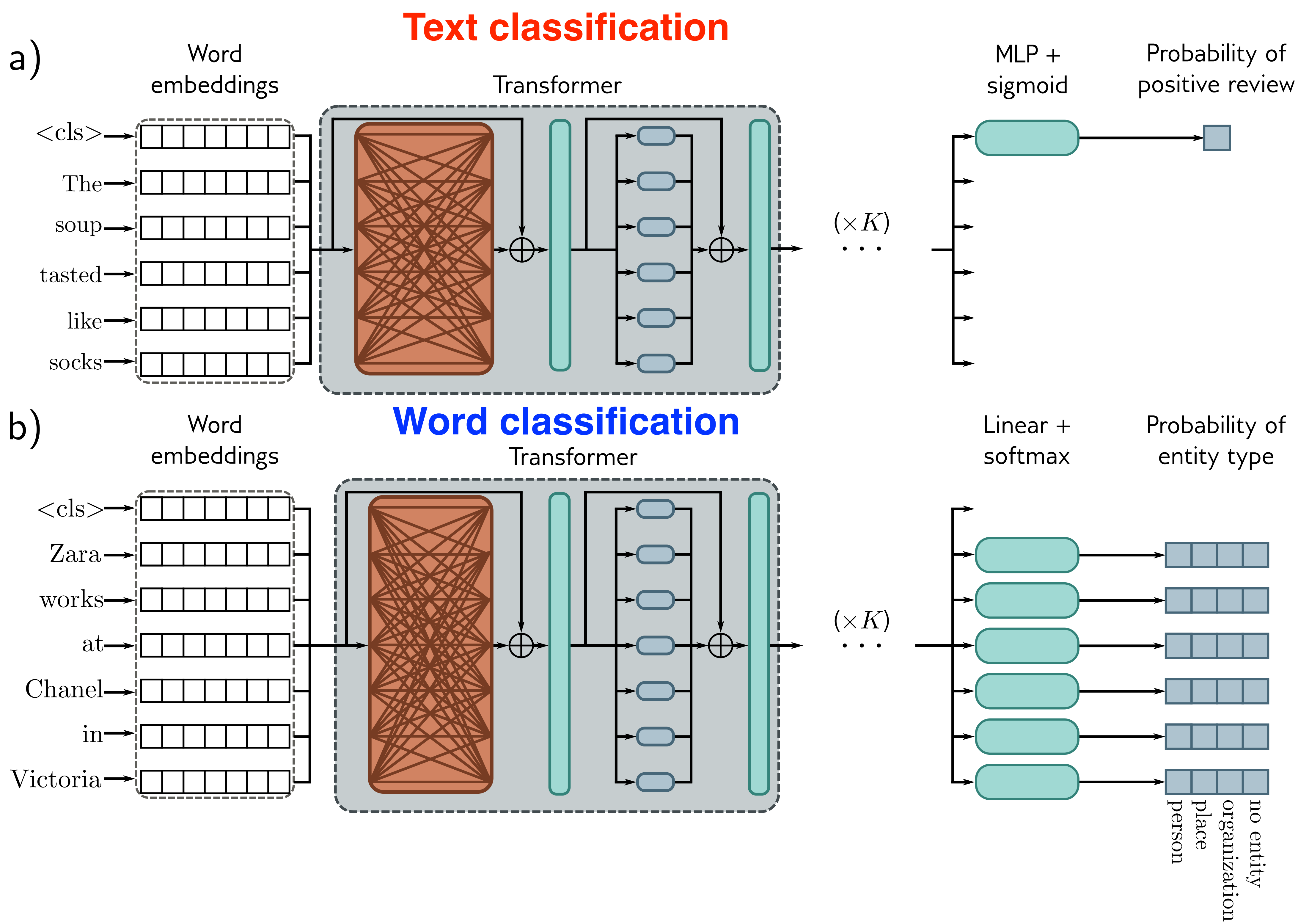


- Predicting missing words forces the transformer model to understand some syntax. For example, red is often found before car or dress than swim. In the above example, train is more likely than lasagna.

Fine-tuning

- In the fine-tuning stage, the model parameters are adjusted to specialize the network to a particular task.
- An extra layer is appended onto the transformer network to convert the output vectors to the desired output format.
- Specific tasks include:
  - **Text classification**: <cls> token is added to the start of each string during pre-training. sentiment analysis, the vector associated with <cls> is mapped to a number & passed through a logistic sigmoid.
  - **Word classification**: e.g., to classify a word into entity types (person, place, organization, or no-entry). Input is mapped to a $E \times 1$ vector where $E =$ entry types, then Softmax for probabilities.
  - **Text span predictions:** A question & a passage from Wikipedia containing the answer are inputs, predicts the text span of answer.

Fine-tuning



Text classification
a)
Word embeddings
Transformer
MLP + sigmoid
Probability of positive review
<cls>
The
soup
tasted
like
socks
$(\times K)$
Word classification
b)
Word embeddings
Transformer
Linear + softmax
Probability of entity type
<cls>
Zara
works
at
Chanel
in
Victoria
$(\times K)$
person
place
organization
no entity

Decoder model example: GPT3

- The basic architecture is similar to the encoder model & comprises a series of transformer layers that operate on learned word embeddings.

- Different goal: to generate the next token in a sequence (and generate a coherent text passage by feeding the sequence back into the model).

- **Autoregressive langauge model**: factors the joint probability of a sequence of observed tokens into an autoregressive sequence.

- Consider e.g.: "It takes great courage to let yourself appear weak."

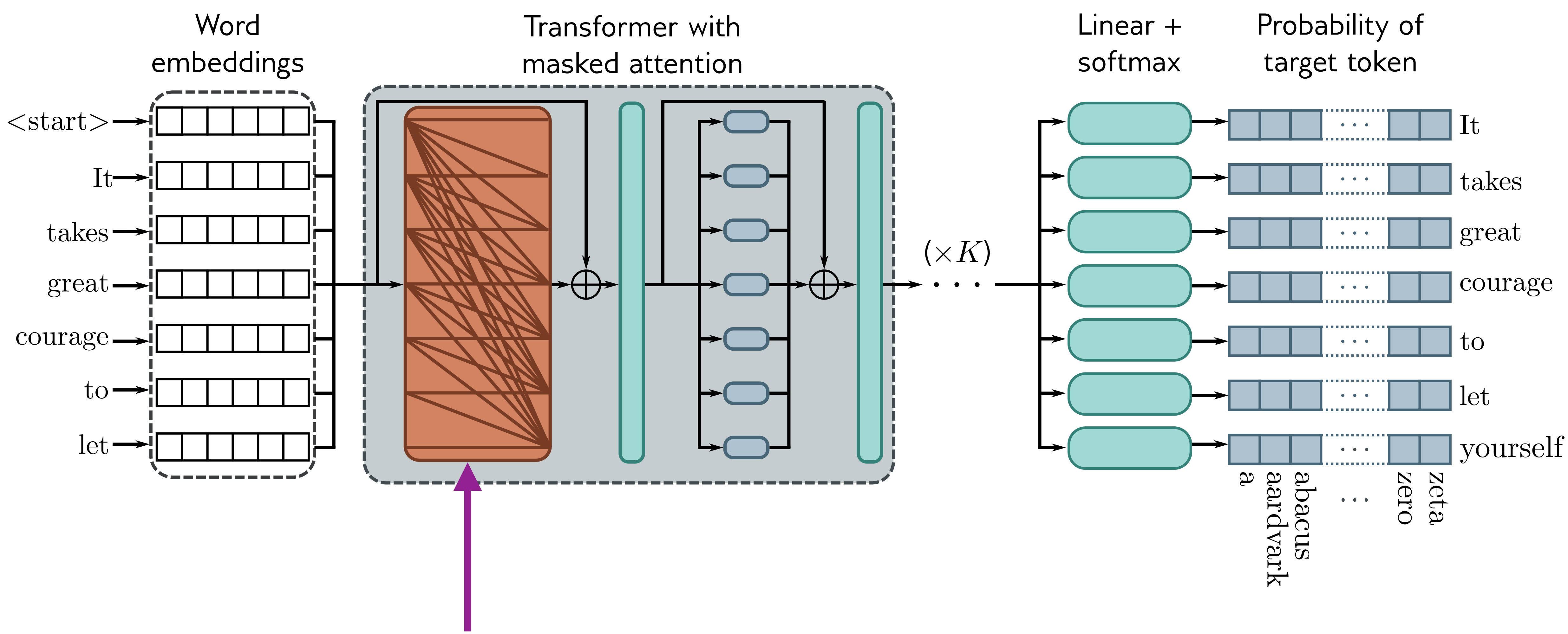
$$
\begin{aligned}
Pr(\text{It takes great courage to let yourself appear weak}) \quad = \\
& Pr(\text{It}) \times Pr(\text{takes}|\text{It}) \times Pr(\text{great}|\text{It takes}) \times Pr(\text{courage}|\text{It takes great}) \times \\
& Pr(\text{to}|\text{It takes great courage}) \times Pr(\text{let}|\text{It takes great courage to}) \times \\
& Pr(\text{yourself}|\text{It takes great courage to let}) \times \\
& Pr(\text{appear}|\text{It takes great courage to let yourself}) \times \\
& Pr(\text{weak}|\text{It takes great courage to let yourself appear}).
\end{aligned}
$$

Generally: $$Pr(t_1, t_2, \ldots, t_N) = Pr(t_1) \prod_{n=2}^{N} Pr(t_n|t_1, \ldots, t_{n-1}).$$

Decoder model example: GPT3

- To train a decoder, we maximize the log probability of the input text under the autoregressive model defined above.
- This poses a problem: if we pass the full sentence, the term computing $\log | Pr(\text{great} | \text{It takes})$ has access to the rest of the sentence.
- The system can cheat rather than learn to predict, and thus will not train properly.
- **Masked self-attention**: setting the dot products with future tokens in the self-attention computation to $-\infty$ before passing through softmax.
- The transformer layers use masked self-attention so that only attention to the current and previous tokens are allowed.
- During training, we aim to maximize the sum of the log probabilities of the next token using a standard multclass cross-entropy loss.

Masked self-attention



attend only to the current and previous tokens

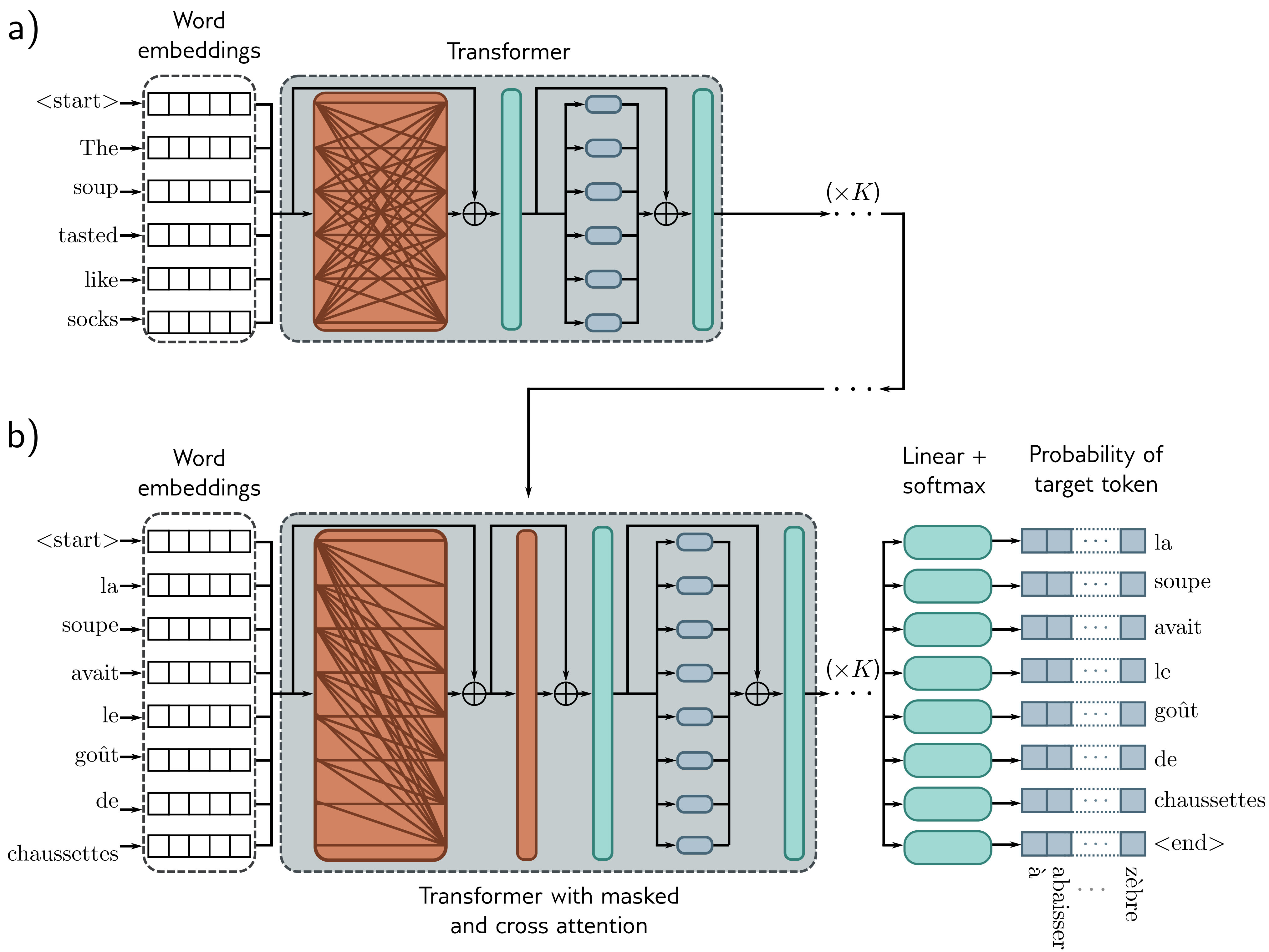
Generating text from a decoder

- The autoregressive language model is a generative model.
- Start with an input sequence of text, beginning with a <start> token.
- The outputs are the probabilities over possible subsequent tokens.
- We can either pick the most likely token or sample from this probability distribution.
- The new extended sequence can be fed back into the decoder network that outputs the probability distribution over the next token.
- Rinse and repeat: we generate large bodies of text.
- The computation is efficient as prior embeddings do not depend on subsequent ones (masked self-attention) and can be recycled.
- Other strategies (instead of greedy search): beam search and top-k sampling, etc.

Encoder-decoder model example: machine translation

- Translation between languages is a sequence-to-sequence task.
- An encoder computes a good rep. of the source sentence.
- A decoder generates the sentence in the target language.
- Consider a encoder-decoder model for English-French translation.
- The encoder receives the sentence in English and process it through a series of transformer layers to create an output rep. for each token.
- During training, the decoder receives the ground truth translation in French and passes it through a series of transformer layers that use masked self-attention and predict the following word at each position.
- However, the decoder layers also attend to the output of the encoder. Each French output word is conditioned on the previous output words and the source English sentence.

Encoder-decoder model example: machine translation



Cross-attention

aka encoder-decoder attention