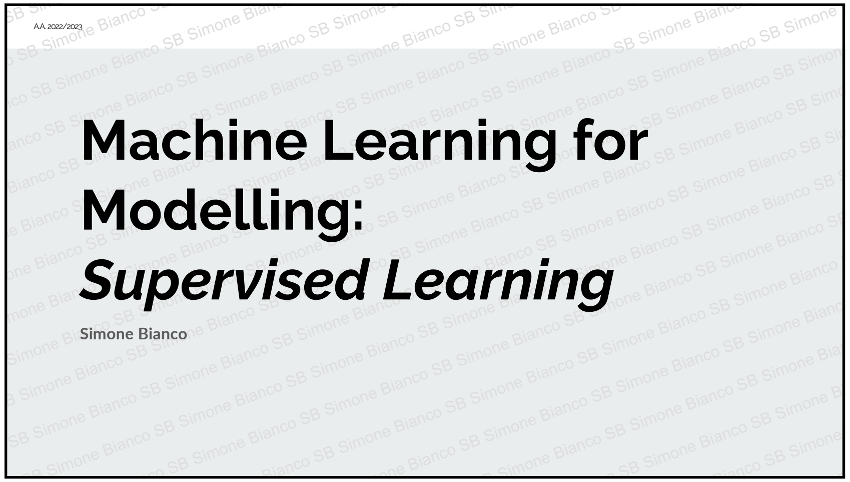
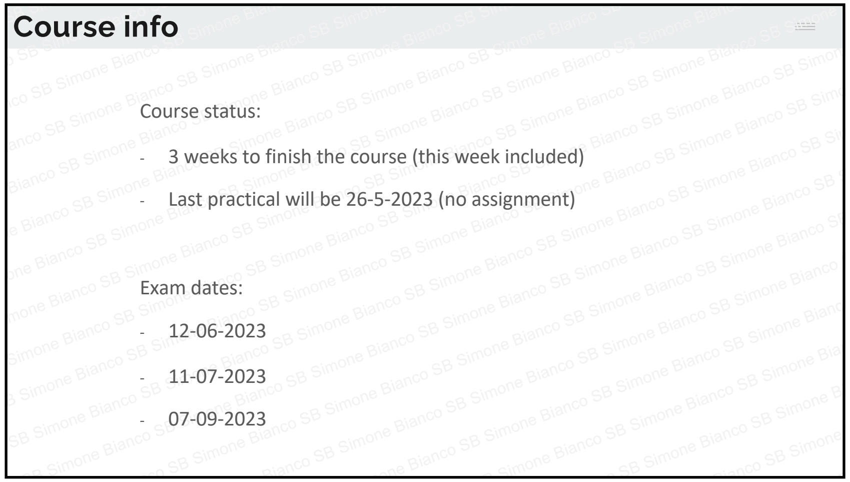


AA 2022/2023

Machine Learning for Modelling: *Supervised Learning*

Simone Bianco

1



Course info

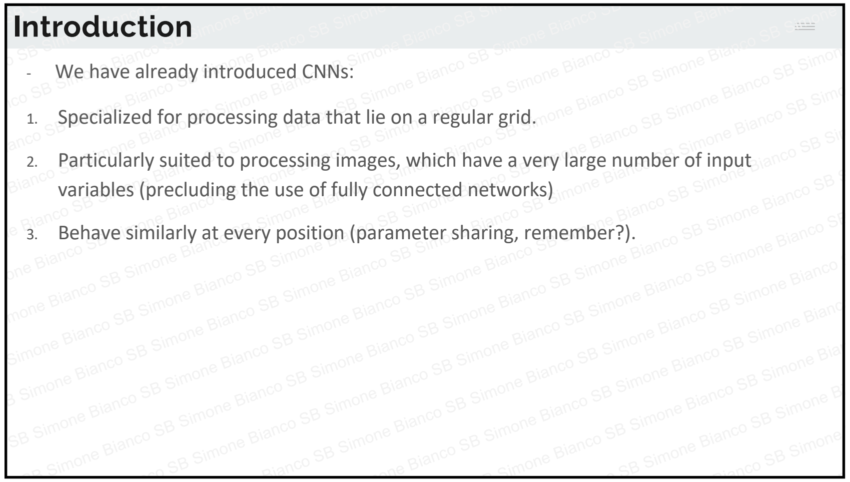
Course status:

- 3 weeks to finish the course (this week included)
- Last practical will be 26-5-2023 (no assignment)

Exam dates:

- 12-06-2023
- 11-07-2023
- 07-09-2023

2

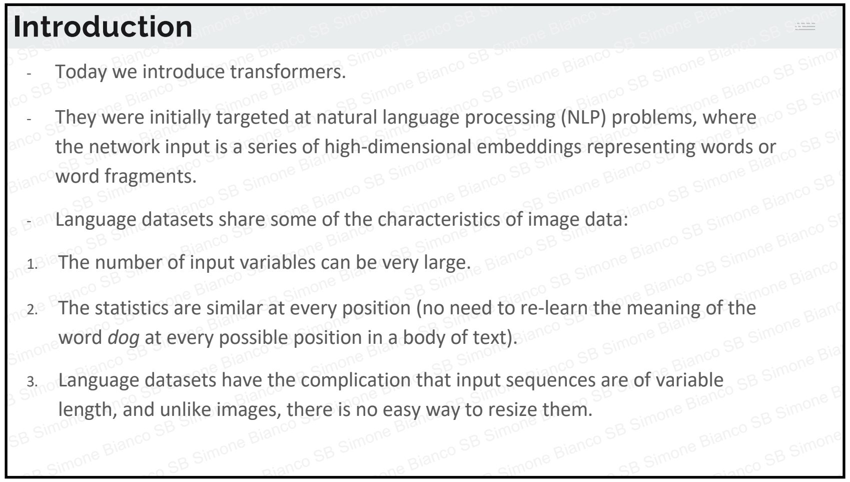


Introduction

- We have already introduced CNNs:

1. Specialized for processing data that lie on a regular grid.
2. Particularly suited to processing images, which have a very large number of input variables (precluding the use of fully connected networks)
3. Behave similarly at every position (parameter sharing, remember?).

3



Introduction

- Today we introduce transformers.
- They were initially targeted at natural language processing (NLP) problems, where the network input is a series of high-dimensional embeddings representing words or word fragments.
- Language datasets share some of the characteristics of image data:

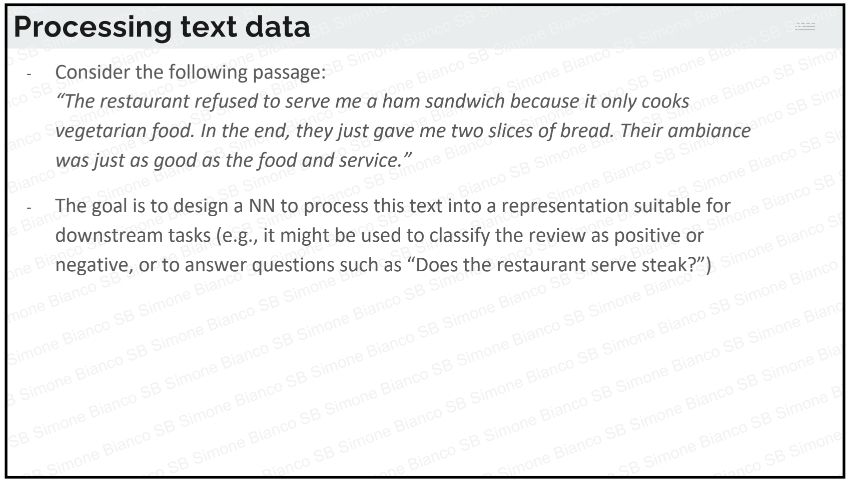
1. The number of input variables can be very large.
2. The statistics are similar at every position (no need to re-learn the meaning of the word *dog* at every possible position in a body of text).
3. Language datasets have the complication that input sequences are of variable length, and unlike images, there is no easy way to resize them.

4

AA 2022/2023

Transformers

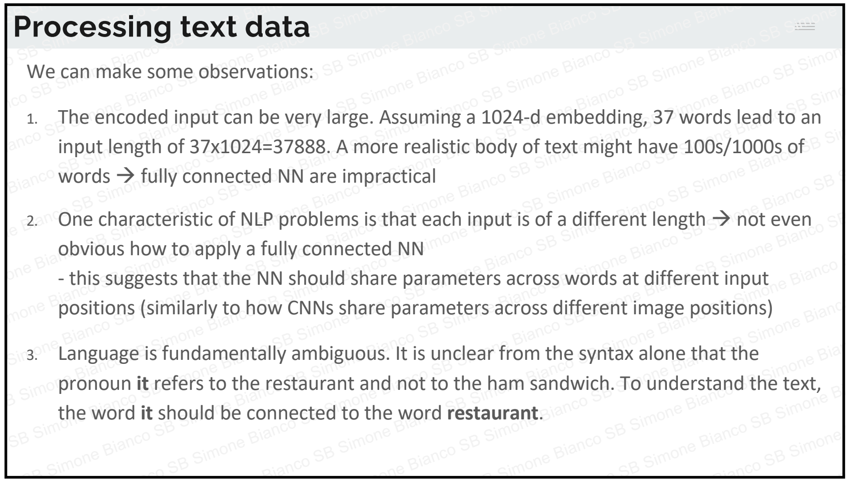
5



Processing text data

- Consider the following passage:
  "The restaurant refused to serve me a ham sandwich because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambiance was just as good as the food and service."
- The goal is to design a NN to process this text into a representation suitable for downstream tasks (e.g., it might be used to classify the review as positive or negative, or to answer questions such as "Does the restaurant serve steak?")

6

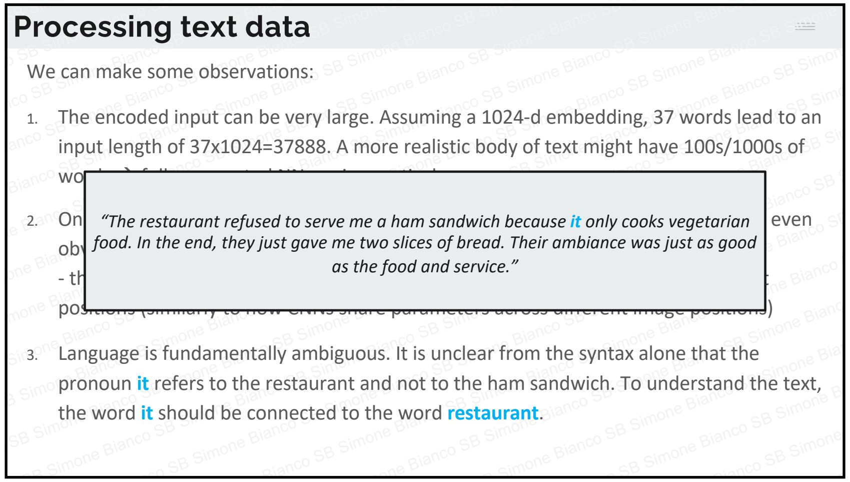


Processing text data

We can make some observations:

1. The encoded input can be very large. Assuming a 1024-d embedding, 37 words lead to an input length of 37x1024=37888. A more realistic body of text might have 100s/1000s of words → fully connected NN are impractical
2. One characteristic of NLP problems is that each input is of a different length → not even obvious how to apply a fully connected NN
   - this suggests that the NN should share parameters across words at different input positions (similarly to how CNNs share parameters across different image positions)
3. Language is fundamentally ambiguous. It is unclear from the syntax alone that the pronoun **it** refers to the restaurant and not to the ham sandwich. To understand the text, the word **it** should be connected to the word **restaurant**.

7



Processing text data

We can make some observations:

1. The encoded input can be very large. Assuming a 1024-d embedding, 37 words lead to an input length of 37x1024=37888. A more realistic body of text might have 100s/1000s of wo
2. On even obv - th positions (similarly to how CNNs share parameters across different image positions)
3. Language is fundamentally ambiguous. It is unclear from the syntax alone that the pronoun **it** refers to the restaurant and not to the ham sandwich. To understand the text, the word **it** should be connected to the word **restaurant**.

*"The restaurant refused to serve me a ham sandwich because **it** only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambiance was just as good as the food and service."*

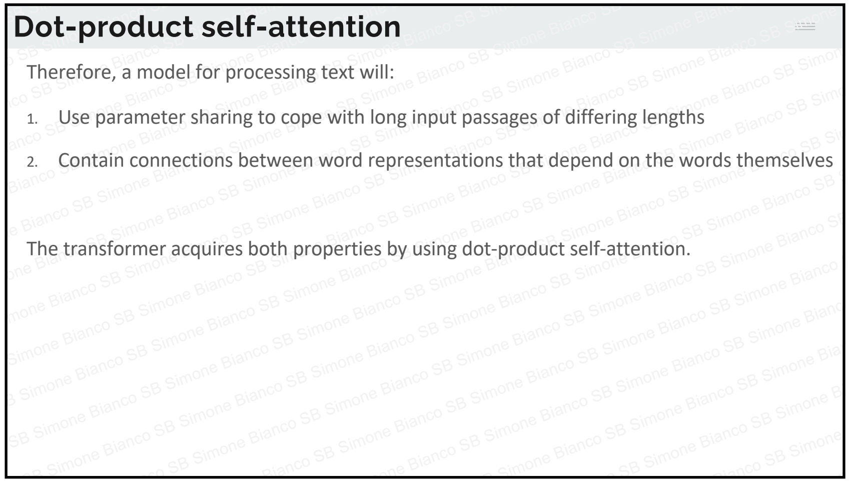
8

AA 2022/2023

Transformers

Dot-product self-attention

9



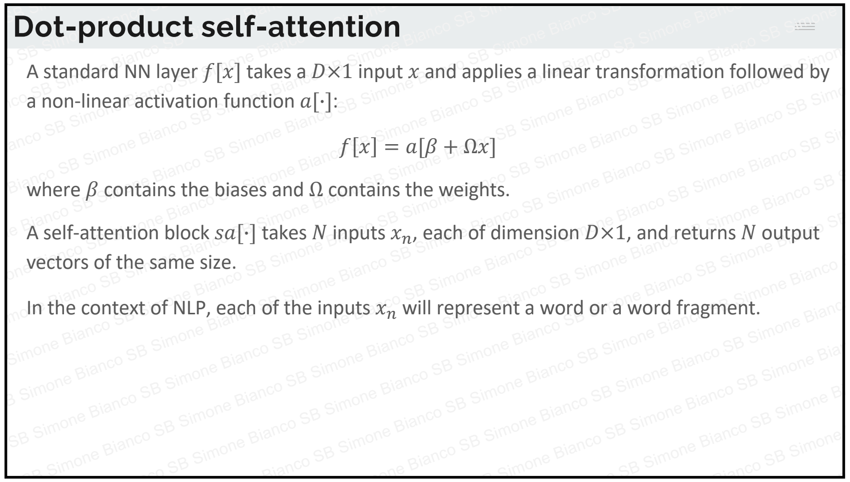
Dot-product self-attention

Therefore, a model for processing text will:

1. Use parameter sharing to cope with long input passages of differing lengths
2. Contain connections between word representations that depend on the words themselves

The transformer acquires both properties by using dot-product self-attention.

10



Dot-product self-attention

A standard NN layer $f[x]$ takes a $D{\times}1$ input x and applies a linear transformation followed by a non-linear activation function $a[\cdot]$:

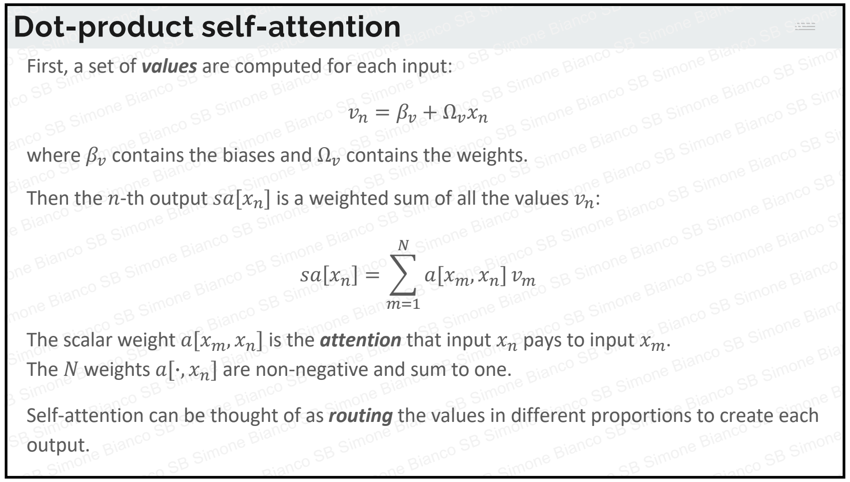
$$f[x] = a[\beta + \Omega x]$$

where β contains the biases and Ω contains the weights.

A self-attention block $sa[\cdot]$ takes N inputs x_n, each of dimension $D{\times}1$, and returns N output vectors of the same size.

In the context of NLP, each of the inputs x_n will represent a word or a word fragment.

11



Dot-product self-attention

First, a set of ***values*** are computed for each input:

$$v_n = \beta_v + \Omega_v x_n$$

where β_v contains the biases and Ω_v contains the weights.

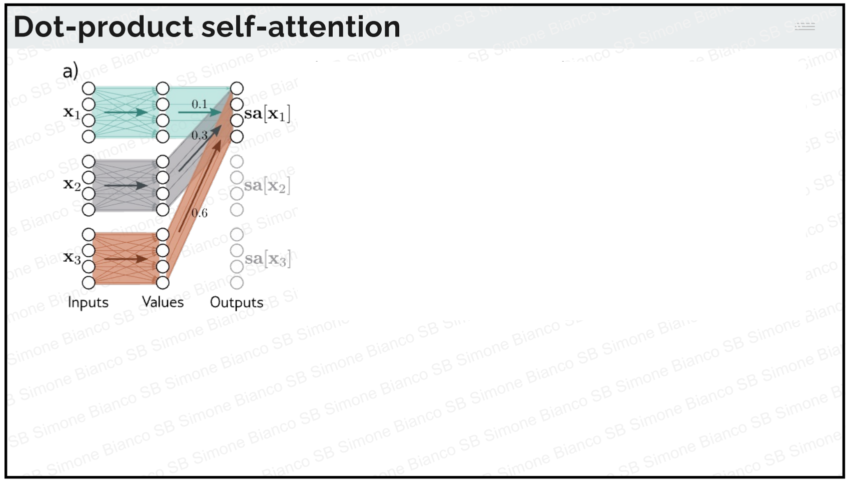
Then the n-th output $sa[x_n]$ is a weighted sum of all the values v_n:

$$sa[x_n] = \sum_{m=1}^{N} a[x_m, x_n]\, v_m$$

The scalar weight $a[x_m, x_n]$ is the ***attention*** that input x_n pays to input x_m.
The N weights $a[\cdot, x_n]$ are non-negative and sum to one.

Self-attention can be thought of as ***routing*** the values in different proportions to create each output.

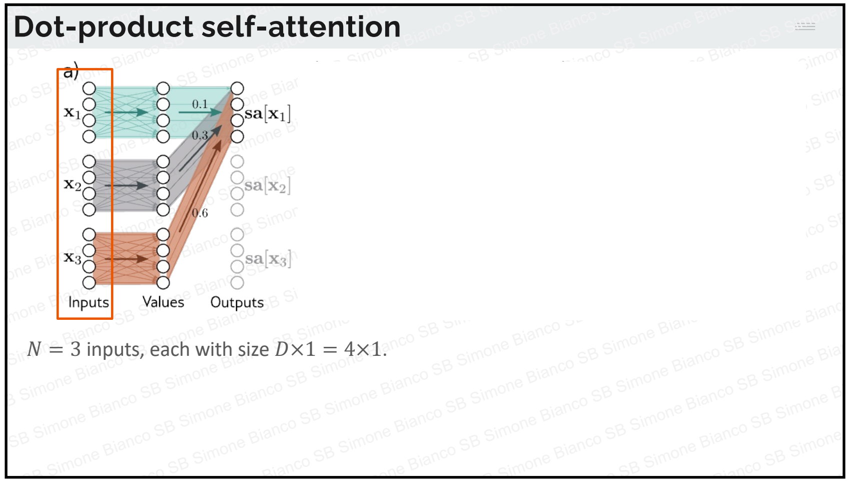
12



Dot-product self-attention

a)
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
0.1
0.3
0.6
$\mathbf{sa}[\mathbf{x}_1]$
$\mathbf{sa}[\mathbf{x}_2]$
$\mathbf{sa}[\mathbf{x}_3]$
Inputs
Values
Outputs

13

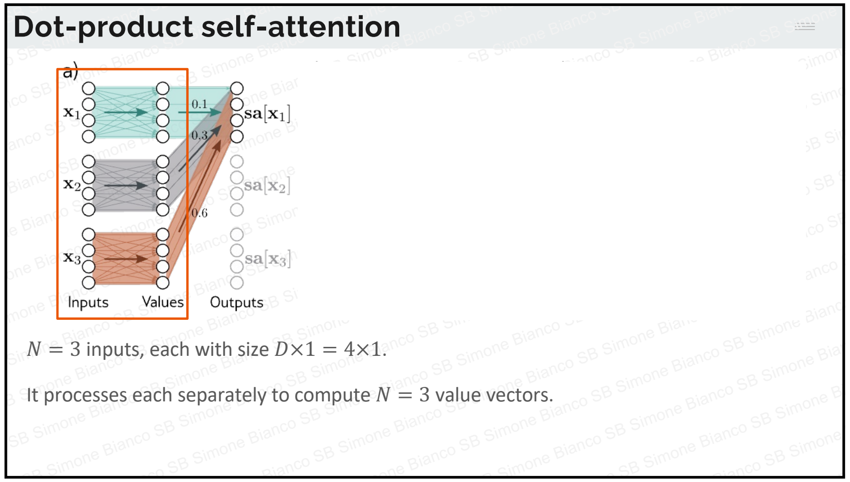


Dot-product self-attention

a)
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
0.1
0.3
0.6
$\mathbf{sa}[\mathbf{x}_1]$
$\mathbf{sa}[\mathbf{x}_2]$
$\mathbf{sa}[\mathbf{x}_3]$
Inputs
Values
Outputs

$N = 3$ inputs, each with size $D{\times}1 = 4{\times}1$.

14



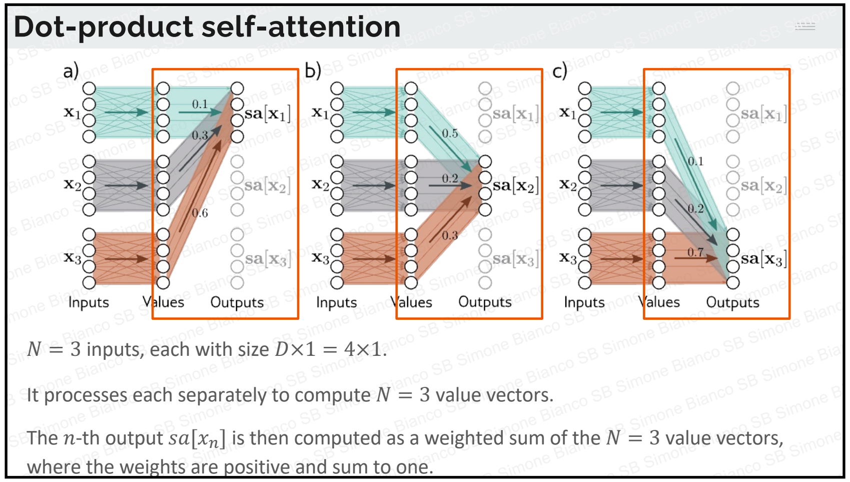
Dot-product self-attention

a)
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
0.1
0.3
0.6
$\mathbf{sa}[\mathbf{x}_1]$
$\mathbf{sa}[\mathbf{x}_2]$
$\mathbf{sa}[\mathbf{x}_3]$
Inputs
Values
Outputs

$N = 3$ inputs, each with size $D{\times}1 = 4{\times}1$.

It processes each separately to compute $N = 3$ value vectors.

15



Dot-product self-attention

a)
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
0.1
0.3
0.6
$\mathbf{sa}[\mathbf{x}_1]$
$\mathbf{sa}[\mathbf{x}_2]$
$\mathbf{sa}[\mathbf{x}_3]$
Inputs
Values
Outputs
b)
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
0.5
0.2
0.3
$\mathbf{sa}[\mathbf{x}_1]$
$\mathbf{sa}[\mathbf{x}_2]$
$\mathbf{sa}[\mathbf{x}_3]$
Inputs
Values
Outputs
c)
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
0.1
0.2
0.7
$\mathbf{sa}[\mathbf{x}_1]$
$\mathbf{sa}[\mathbf{x}_2]$
$\mathbf{sa}[\mathbf{x}_3]$
Inputs
Values
Outputs

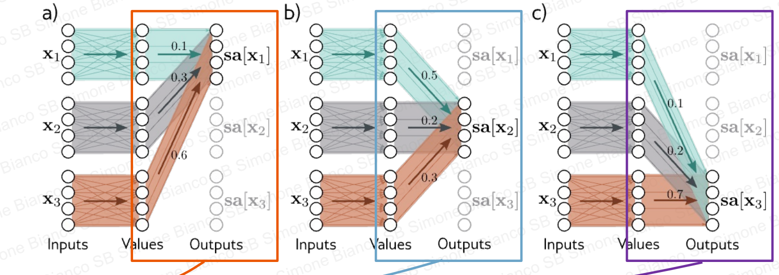
$N = 3$ inputs, each with size $D{\times}1 = 4{\times}1$.

It processes each separately to compute $N = 3$ value vectors.

The n-th output $sa[x_n]$ is then computed as a weighted sum of the $N = 3$ value vectors, where the weights are positive and sum to one.

16

Dot-product self-attention



In particular, we have:

$a[x_1, x_1] = 0.1$	$a[x_1, x_2] = 0.5$	$a[x_1, x_3] = 0.1$
$a[x_2, x_1] = 0.3$	$a[x_2, x_2] = 0.2$	$a[x_2, x_3] = 0.2$
$a[x_3, x_1] = 0.6$	$a[x_3, x_2] = 0.3$	$a[x_3, x_3] = 0.7$

Computing and weighting values

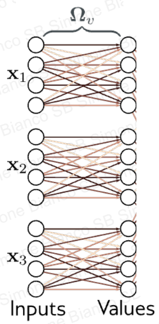
To compute the values, the same weights Ω_v (with size $D{\times}D$) and biases β_v (with size $D{\times}1$) are applied to each input input x_n (with size $D{\times}1$):

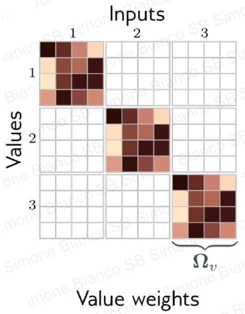
$$v_n = \beta_v + \Omega_v x_n$$

This computation scales linearly with the sequence length N so it requires less parameters than a FC NN connecting all DN inputs to the DN outputs.

The value computation can be thus viewed as a sparse matrix multiplication with shared parameters.

Computing and weighting values





Computing and weighting values

To compute the values, the same weights Ω_v (with size $D{\times}D$) and biases β_v (with size $D{\times}1$) are applied to each input input x_n (with size $D{\times}1$):

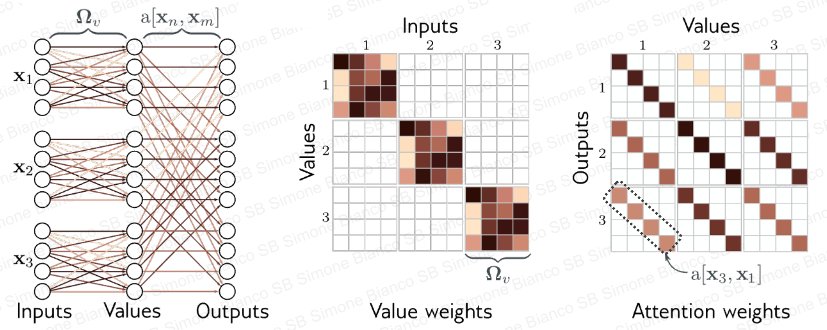
$$v_n = \beta_v + \Omega_v x_n$$

This computation scales linearly with the sequence length N so it requires less parameters than a FC NN connecting all DN inputs to the DN outputs.

The value computation can be thus viewed as a sparse matrix multiplication with shared parameters.

The attention weights $a[x_m, x_n]$ combine the values from different inputs. They are also sparse, since there is only one weight for each ordered pair of inputs (x_m, x_n) regardless of the size of these inputs.

Computing and weighting values



22

Computing and weighting values

To compute the values, the same weights Ω_v (with size $D\times D$) and biases β_v (with size $D\times 1$) are applied to each input input x_n (with size $D\times 1$):

$$v_n = \beta_v + \Omega_v x_n$$

This computation scales linearly with the sequence length N so it requires less parameters than a FC NN connecting all DN inputs to the DN outputs.

The value computation can be thus viewed as a sparse matrix multiplication with shared parameters.

The attention weights $a[x_m, x_n]$ combine the values from different inputs. They are also sparse, since there is only one weight for each ordered pair of inputs (x_m, x_n) regardless of the size of these inputs.

The number of attention weights has a quadratic dependence on the sequence length N but it is independent of the length D of each input x_n.

23

Computing attention weights

We have seen that the outputs result from **two chained linear transformations**:

- The value vectors are computed independently from each input,
- and these vectors are linearly combined by the attention weights.

However, the overall self-attention is **nonlinear** because the attention weights are themselves nonlinear functions of the input.

This is an example of a hypernetwork, where one network branch computes the weights of another branch.

25

Computing attention weights

To compute the attention, we apply two more linear transformations to the inputs:

$$q_n = \beta_q + \Omega_q x_n$$
$$k_n = \beta_k + \Omega_k x_n$$

where q_n and k_n are referred to as the **queries** and the **keys.**

We then compute dot products between the queries and the keys followed by a softmax:

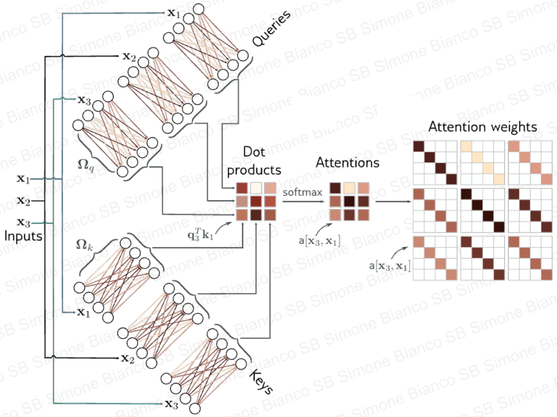
$$a[x_m, x_n] = softmax_m[k_*^T q_n] = \frac{\exp[k_m^T q_n]}{\sum_{m'=1}^{N} \exp[k_{m'}^T q_n]}$$

so, for each x_n they are positive and sum to one.

For obvious reasons, this is known as **dot-product self-attention**.

26

Computing attention weights



27

Computing attention weights

The names “queries” and “keys” were inherited from the field of information retrieval and have the following interpretation:

- the dot product operation returns a measure of similarity between its inputs, so the weights $a[x_*, x_n]$ depend on the relative similarities between each query and the keys.
- The softmax function means that we can think of the key vectors as “competing” with one another to contribute to the final result.

The queries and keys must have the same dimensions.

However, these can differ from the dimension of the values, which is usually the same size as the input so that the representation does not change size.

28

Self-attention summary

The n-th output is a weighted sum of the same linear transformation $v_* = \beta_v + \Omega_v x_*$ applied to all the inputs, where these attention weights are positive and sum to one.

The weights depend on a measure of similarity between input x_n and the other inputs.

There is no activation function, but the mechanism is nonlinear due to the dot-product and a softmax operation used to compute the attention weights.

This mechanism fulfills the initial requirements:

1. There is a single shared set of parameters $\phi = \{\beta_v, \Omega_v, \beta_q, \Omega_q, \beta_k, \Omega_k\}$. This is independent of the number of inputs N, so the networks can be applied to difference sequence lengths.
2. There are connections between the inputs (words), and the strength of these connections depends on the inputs themselves via the attention weights.

29

Matrix form

The previous computation can be written in a compact form if the N inputs x_n form the columns of the $D{\times}N$ matrix X.

The values, queries, and keys can be computed as:

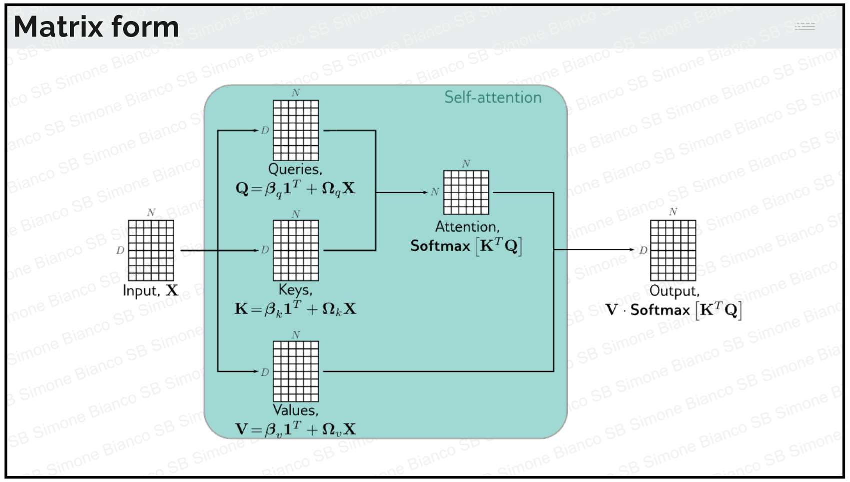
$$V[X] = \beta_v 1^T + \Omega_v X$$
$$Q[X] = \beta_q 1^T + \Omega_q X$$
$$K[X] = \beta_k 1^T + \Omega_k X$$

The self-attention computation is then:

$$Sa[X] = V[X] \cdot Softmax[K[X]^T Q[X]] = V \cdot Softmax[K^T Q]$$

where softmax is independently applied on the columns of its input.

30



Matrix form

Self-attention
N
D
Input, $\mathbf{X}$
Queries,
$\mathbf{Q}=\boldsymbol{\beta}_q\mathbf{1}^T+\boldsymbol{\Omega}_q\mathbf{X}$
Keys,
$\mathbf{K}=\boldsymbol{\beta}_k\mathbf{1}^T+\boldsymbol{\Omega}_k\mathbf{X}$
Values,
$\mathbf{V}=\boldsymbol{\beta}_v\mathbf{1}^T+\boldsymbol{\Omega}_v\mathbf{X}$
Attention,
Softmax $[\mathbf{K}^T\mathbf{Q}]$
Output,
$\mathbf{V}\cdot$ Softmax $[\mathbf{K}^T\mathbf{Q}]$

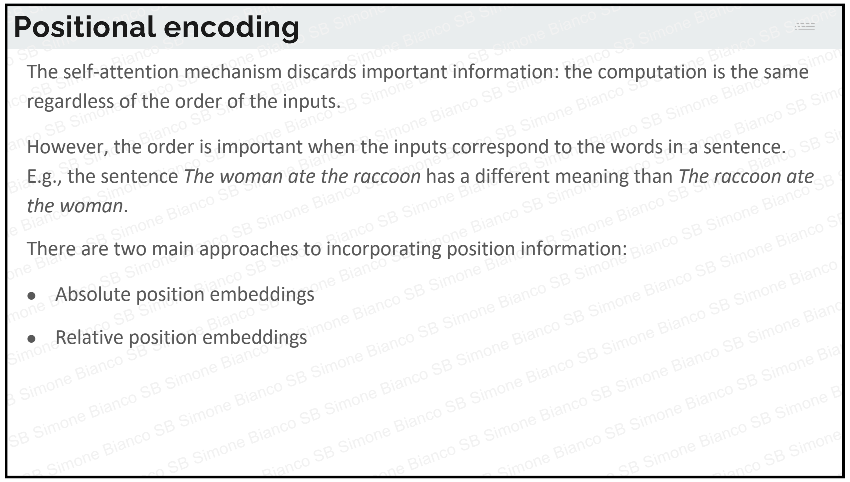
31

AA 2022/2023

Transformers

Extensions to dot-product self-attention

32



Positional encoding

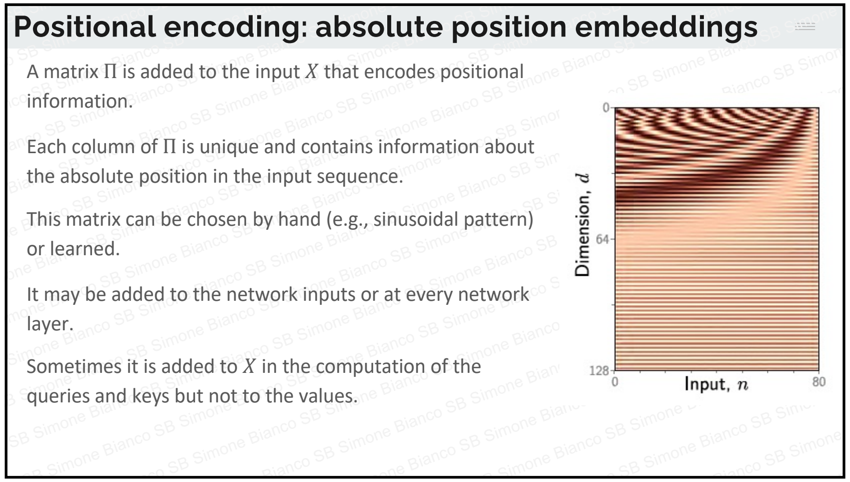
The self-attention mechanism discards important information: the computation is the same regardless of the order of the inputs.

However, the order is important when the inputs correspond to the words in a sentence. E.g., the sentence *The woman ate the raccoon* has a different meaning than *The raccoon ate the woman*.

There are two main approaches to incorporating position information:

- Absolute position embeddings
- Relative position embeddings

33



Positional encoding: absolute position embeddings

A matrix Π is added to the input X that encodes positional information.

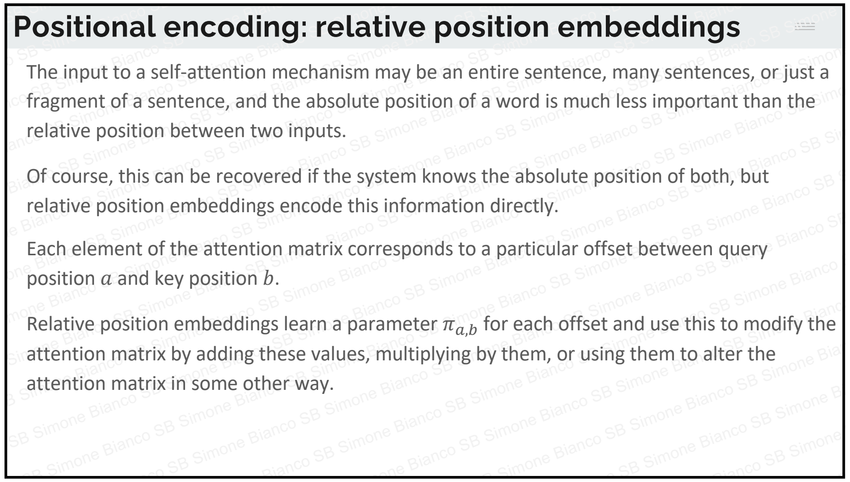
Each column of Π is unique and contains information about the absolute position in the input sequence.

This matrix can be chosen by hand (e.g., sinusoidal pattern) or learned.

It may be added to the network inputs or at every network layer.

Sometimes it is added to X in the computation of the queries and keys but not to the values.

34



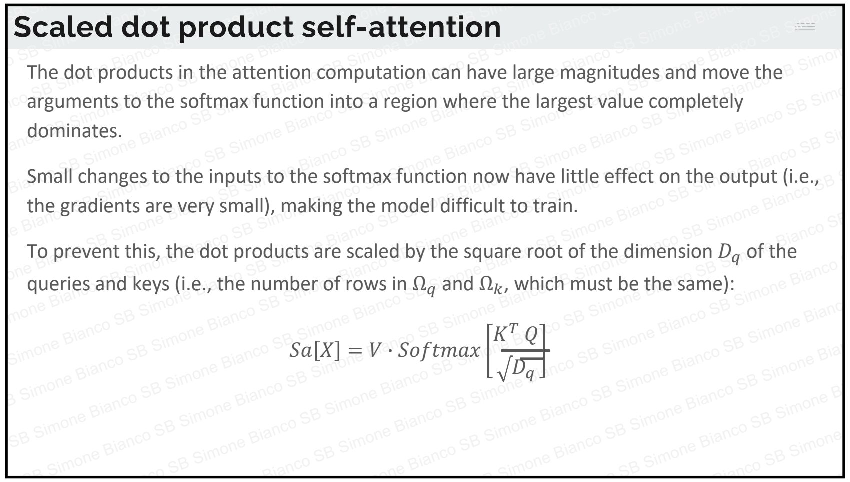
Positional encoding: relative position embeddings

The input to a self-attention mechanism may be an entire sentence, many sentences, or just a fragment of a sentence, and the absolute position of a word is much less important than the relative position between two inputs.

Of course, this can be recovered if the system knows the absolute position of both, but relative position embeddings encode this information directly.

Each element of the attention matrix corresponds to a particular offset between query position a and key position b.

Relative position embeddings learn a parameter $\pi_{a,b}$ for each offset and use this to modify the attention matrix by adding these values, multiplying by them, or using them to alter the attention matrix in some other way.



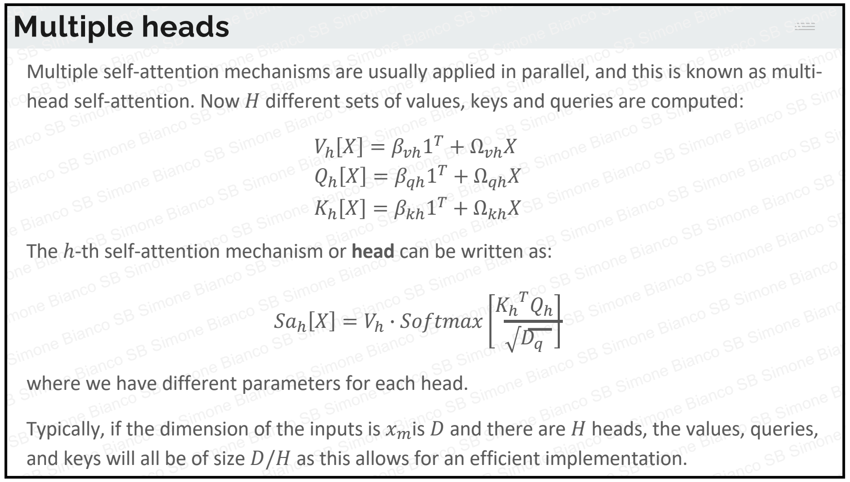
Scaled dot product self-attention

The dot products in the attention computation can have large magnitudes and move the arguments to the softmax function into a region where the largest value completely dominates.

Small changes to the inputs to the softmax function now have little effect on the output (i.e., the gradients are very small), making the model difficult to train.

To prevent this, the dot products are scaled by the square root of the dimension D_q of the queries and keys (i.e., the number of rows in Ω_q and Ω_k, which must be the same):

$$Sa[X] = V \cdot Softmax\left[\frac{K^T\,Q}{\sqrt{D_q}}\right]$$



Multiple heads

Multiple self-attention mechanisms are usually applied in parallel, and this is known as multi-head self-attention. Now H different sets of values, keys and queries are computed:

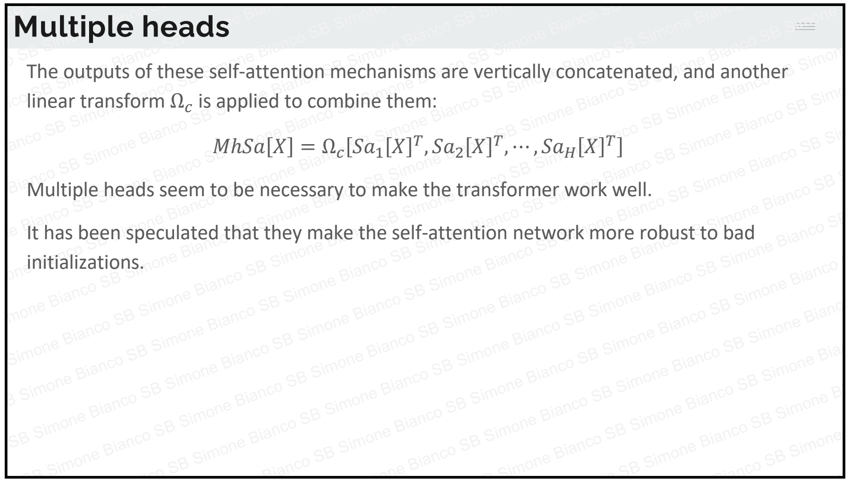
$$V_h[X] = \beta_{vh}1^T + \Omega_{vh}X$$
$$Q_h[X] = \beta_{qh}1^T + \Omega_{qh}X$$
$$K_h[X] = \beta_{kh}1^T + \Omega_{kh}X$$

The h-th self-attention mechanism or **head** can be written as:

$$Sa_h[X] = V_h \cdot Softmax\left[\frac{{K_h}^T Q_h}{\sqrt{D_q}}\right]$$

where we have different parameters for each head.

Typically, if the dimension of the inputs is $x_m$is D and there are H heads, the values, queries, and keys will all be of size D/H as this allows for an efficient implementation.



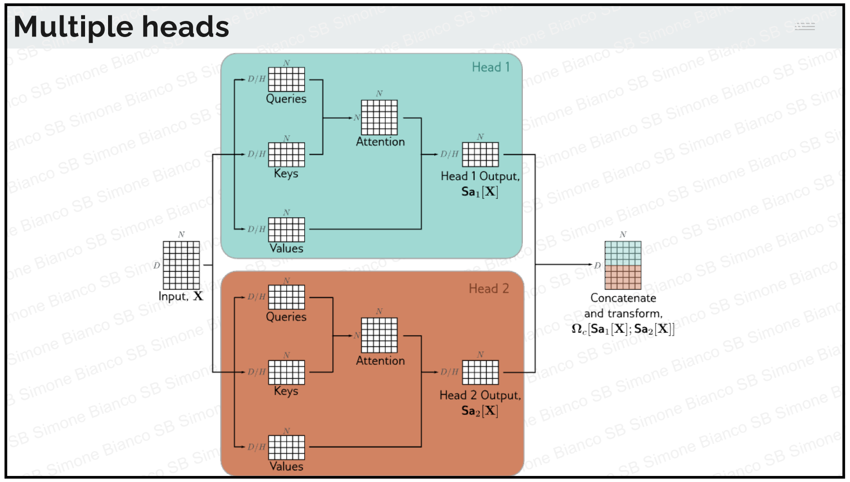
Multiple heads

The outputs of these self-attention mechanisms are vertically concatenated, and another linear transform Ω_c is applied to combine them:

$$MhSa[X] = \Omega_c[Sa_1[X]^T, Sa_2[X]^T, \cdots, Sa_H[X]^T]$$

Multiple heads seem to be necessary to make the transformer work well.

It has been speculated that they make the self-attention network more robust to bad initializations.



Multiple heads

Head 1
Head 2
Input, $\mathbf{X}$
D
N
D/H
Queries
Keys
Values
Attention
Head 1 Output, $\mathbf{Sa}_1[\mathbf{X}]$
Head 2 Output, $\mathbf{Sa}_2[\mathbf{X}]$
Concatenate and transform, $\mathbf{\Omega}_c[\mathbf{Sa}_1[\mathbf{X}]; \mathbf{Sa}_2[\mathbf{X}]]$

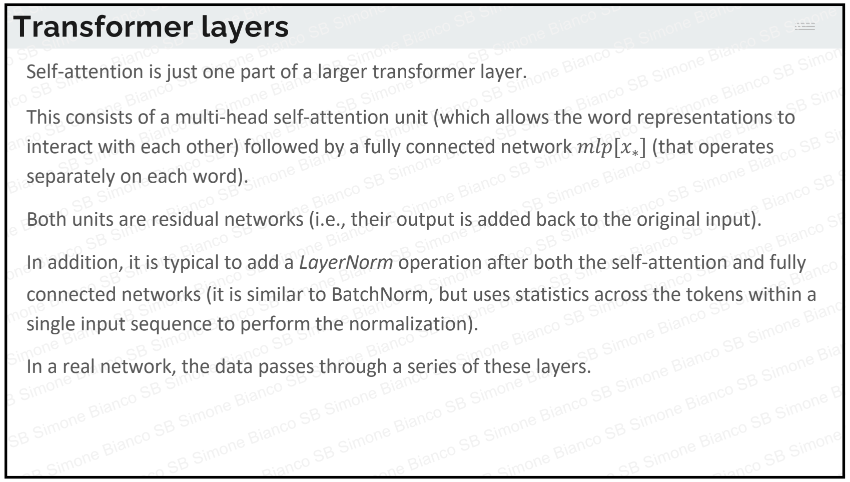
39

AA 2022/2023

Transformers

Transformer layers

40



Transformer layers

Self-attention is just one part of a larger transformer layer.

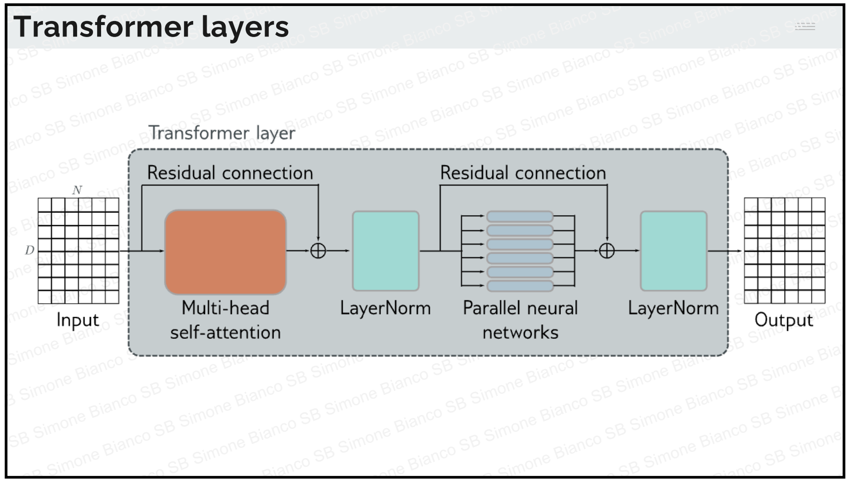
This consists of a multi-head self-attention unit (which allows the word representations to interact with each other) followed by a fully connected network $mlp[x_*]$ (that operates separately on each word).

Both units are residual networks (i.e., their output is added back to the original input).

In addition, it is typical to add a *LayerNorm* operation after both the self-attention and fully connected networks (it is similar to BatchNorm, but uses statistics across the tokens within a single input sequence to perform the normalization).

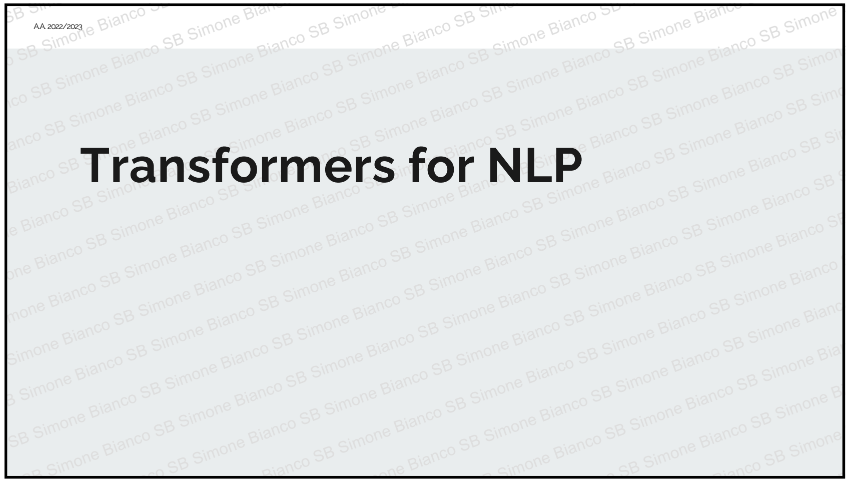
In a real network, the data passes through a series of these layers.

41



Transformer layers

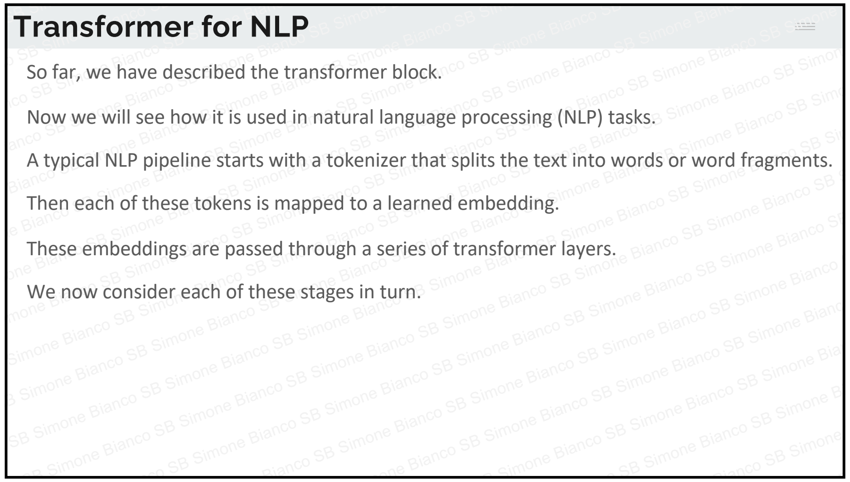
42



AA 2022/2023

Transformers for NLP

43



Transformer for NLP

So far, we have described the transformer block.

Now we will see how it is used in natural language processing (NLP) tasks.

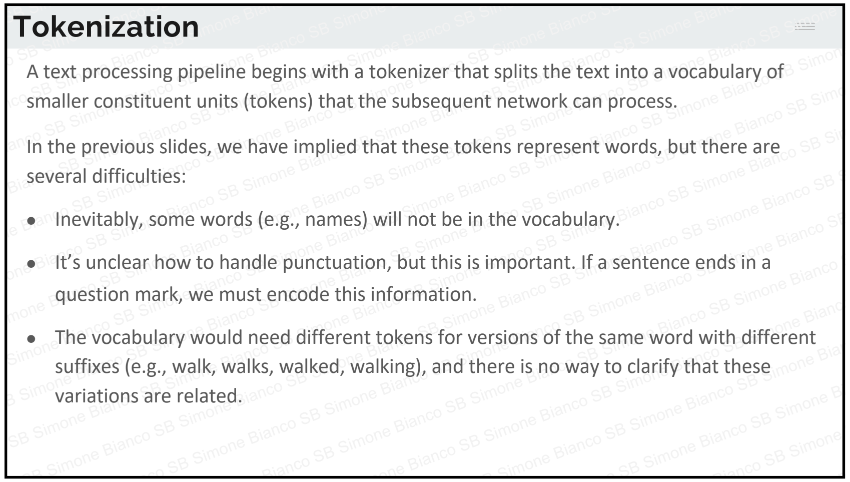
A typical NLP pipeline starts with a tokenizer that splits the text into words or word fragments.

Then each of these tokens is mapped to a learned embedding.

These embeddings are passed through a series of transformer layers.

We now consider each of these stages in turn.

44



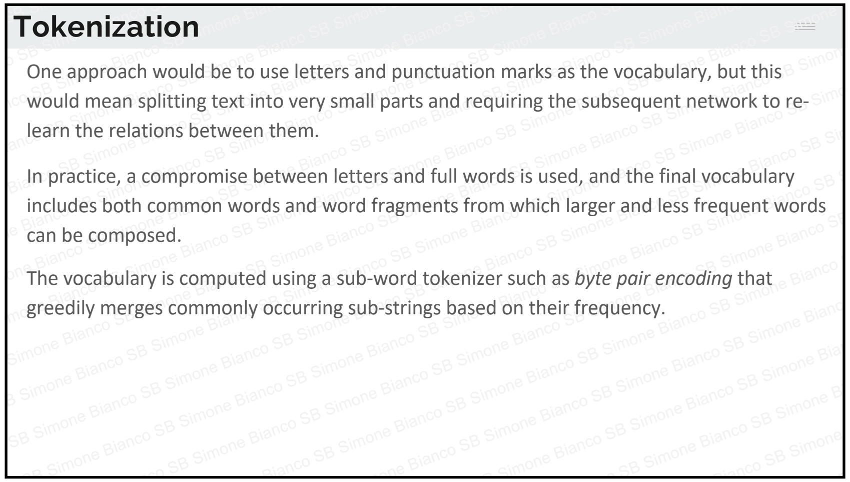
Tokenization

A text processing pipeline begins with a tokenizer that splits the text into a vocabulary of smaller constituent units (tokens) that the subsequent network can process.

In the previous slides, we have implied that these tokens represent words, but there are several difficulties:

- Inevitably, some words (e.g., names) will not be in the vocabulary.
- It's unclear how to handle punctuation, but this is important. If a sentence ends in a question mark, we must encode this information.
- The vocabulary would need different tokens for versions of the same word with different suffixes (e.g., walk, walks, walked, walking), and there is no way to clarify that these variations are related.

45



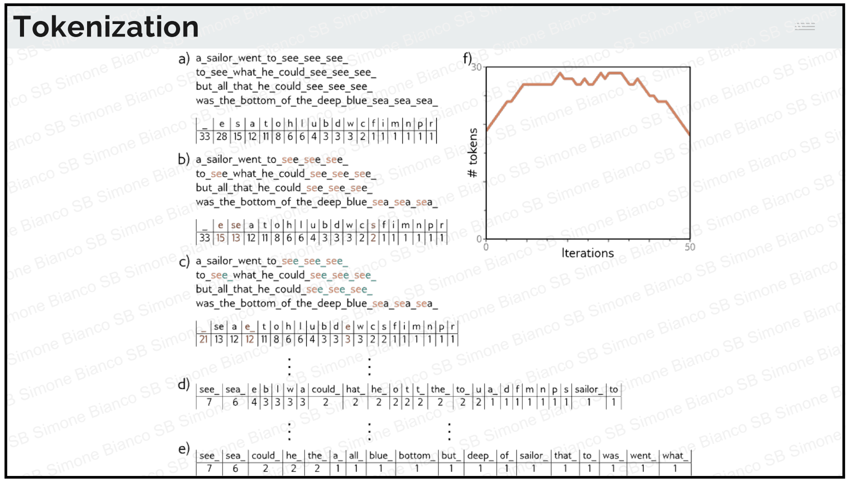
Tokenization

One approach would be to use letters and punctuation marks as the vocabulary, but this would mean splitting text into very small parts and requiring the subsequent network to re-learn the relations between them.

In practice, a compromise between letters and full words is used, and the final vocabulary includes both common words and word fragments from which larger and less frequent words can be composed.

The vocabulary is computed using a sub-word tokenizer such as *byte pair encoding* that greedily merges commonly occurring sub-strings based on their frequency.

46



Tokenization

a) a_sailor_went_to_see_see_see_
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

b) a_sailor_went_to_see_see_see_
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

c) a_sailor_went_to_see_see_see_
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

⋮

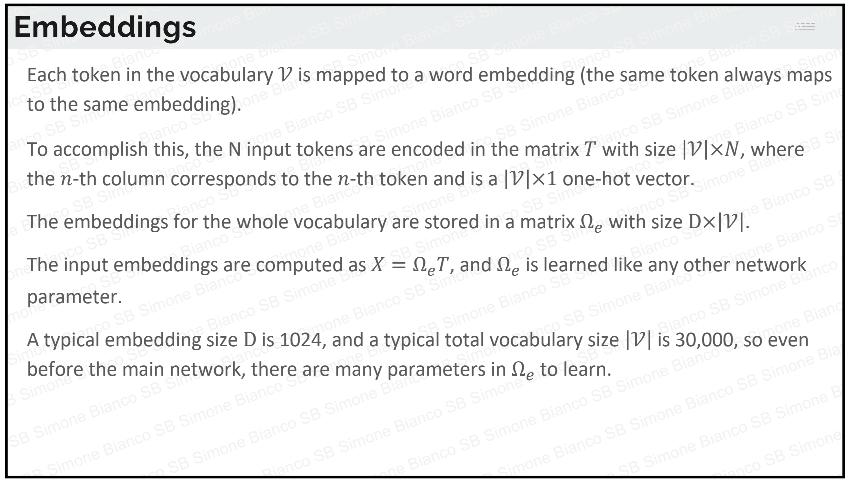
d)

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

⋮

e)

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



Embeddings

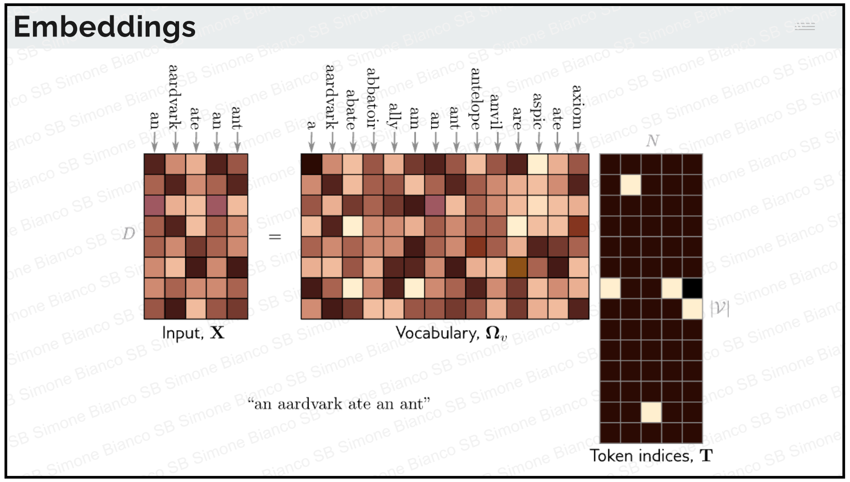
Each token in the vocabulary $\mathcal{V}$ is mapped to a word embedding (the same token always maps to the same embedding).

To accomplish this, the N input tokens are encoded in the matrix T with size $|\mathcal{V}|\times N$, where the n-th column corresponds to the n-th token and is a $|\mathcal{V}|\times 1$ one-hot vector.

The embeddings for the whole vocabulary are stored in a matrix Ω_e with size D×$|\mathcal{V}|$.

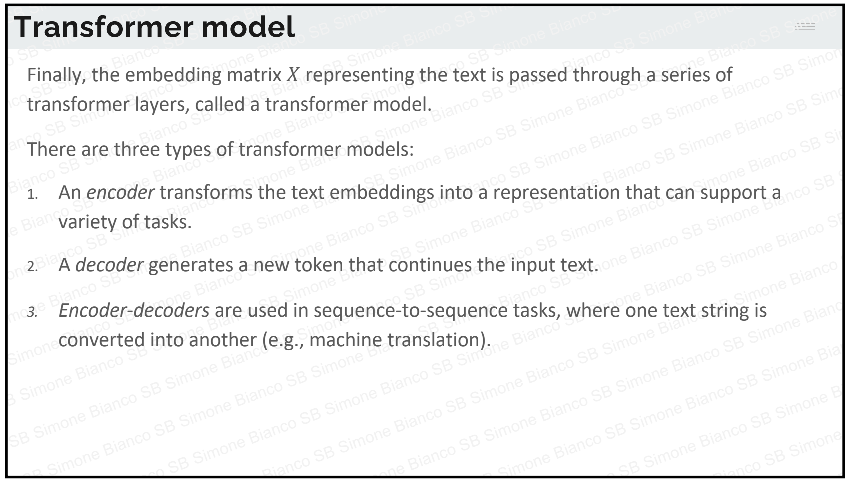
The input embeddings are computed as $X = \Omega_e T$, and Ω_e is learned like any other network parameter.

A typical embedding size D is 1024, and a typical total vocabulary size $|\mathcal{V}|$ is 30,000, so even before the main network, there are many parameters in Ω_e to learn.



Embeddings

“an aardvark ate an ant”

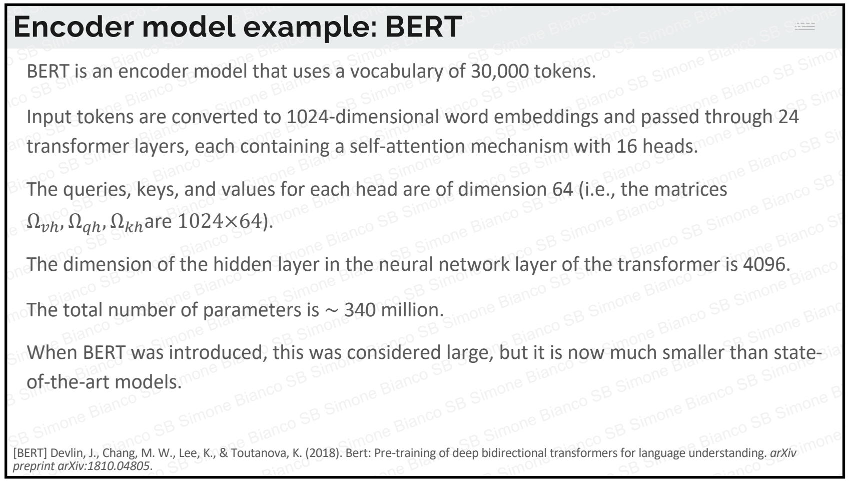


Transformer model

Finally, the embedding matrix X representing the text is passed through a series of transformer layers, called a transformer model.

There are three types of transformer models:

1. An *encoder* transforms the text embeddings into a representation that can support a variety of tasks.
2. A *decoder* generates a new token that continues the input text.
3. *Encoder-decoders* are used in sequence-to-sequence tasks, where one text string is converted into another (e.g., machine translation).



Encoder model example: BERT

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 containing a self-attention mechanism with 16 heads.

The queries, keys, and values for each head are of dimension 64 (i.e., the matrices $\Omega_{vh}, \Omega_{qh}, \Omega_{kh}$ are 1024×64).

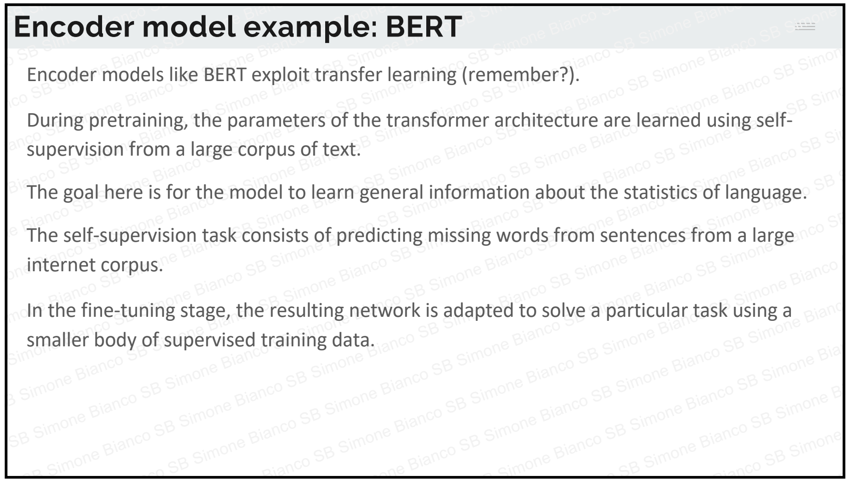
The dimension of the hidden layer in the neural network layer of the transformer is 4096.

The total number of parameters is ~ 340 million.

When BERT was introduced, this was considered large, but it is now much smaller than state-of-the-art models.

[BERT] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

51



Encoder model example: BERT

Encoder models like BERT exploit transfer learning (remember?).

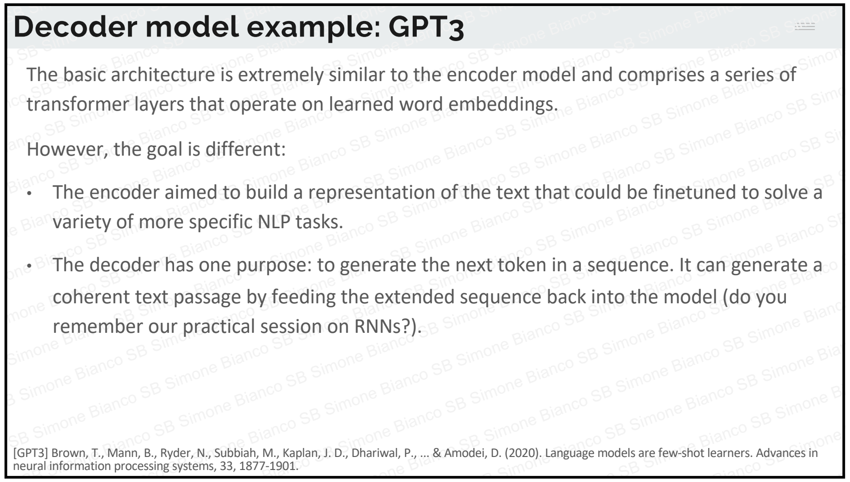
During pretraining, the parameters of the transformer architecture are learned using self-supervision from a large corpus of text.

The goal here is for the model to learn general information about the statistics of language.

The self-supervision task consists of predicting missing words from sentences from a large internet corpus.

In the fine-tuning stage, the resulting network is adapted to solve a particular task using a smaller body of supervised training data.

52



Decoder model example: GPT3

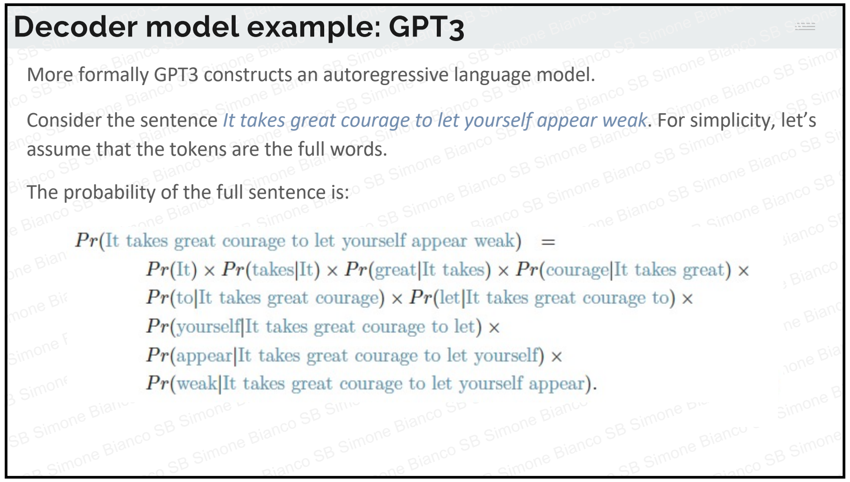
The basic architecture is extremely similar to the encoder model and comprises a series of transformer layers that operate on learned word embeddings.

However, the goal is different:

- The encoder aimed to build a representation of the text that could be finetuned to solve a variety of more specific NLP tasks.
- The decoder has one purpose: to generate the next token in a sequence. It can generate a coherent text passage by feeding the extended sequence back into the model (do you remember our practical session on RNNs?).

[GPT3] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.

53



Decoder model example: GPT3

More formally GPT3 constructs an autoregressive language model.

Consider the sentence *It takes great courage to let yourself appear weak*. For simplicity, let's assume that the tokens are the full words.

The probability of the full sentence is:

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

54

Decoder model example: GPT3

More formally, an autoregressive model factors the joint probability $\Pr(t_1, t_2, \cdots, t_N)$ of the N observed tokens into an autoregressive sequence:

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

The autoregressive language model is a *generative model*.

Since it defines a probability model over text sequences, it can be used to sample new examples of plausible text.

To generate from the model, we start with an input sequence of text (which might be just a special <start> token indicating the beginning of the sequence) and feed this into the network, which then outputs the probabilities over possible subsequent tokens.

55

Decoder model example: GPT3

We can then 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. By repeating this process, we can generate large bodies of text.

GPT3 applies these ideas on a massive scale: The sequence lengths are 2048 tokens long, and since multiple spans of 2048 tokens are processed at once, the batch size is 3.2 million tokens.

There are 96 transformer layers (some of which implement a sparse version of attention), each processing a word embedding of size 12288.

There are 96 heads in the self-attention layers, and the value, query, and key dimension is 128.

It is trained with 300 billion tokens and contains 175 billion parameters.

56

Encoder-Decoder mdl example: machine translation

Translation between languages is an example of a sequence-to-sequence task.

This requires an encoder (to compute a good representation of the source sentence) and a decoder (to generate the sentence in the target language).

This task can be tackled using an encoder-decoder model.

Consider the example of translating from English to French:

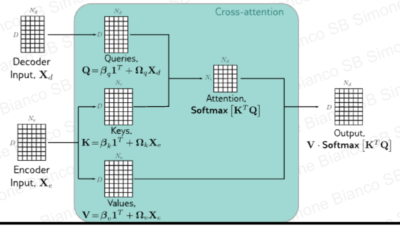
- The encoder receives the sentence in English and processes it through a series of transformer layers to create an output representation for each token.
- During training, the decoder receives the ground truth translation in French and passes it through a series of transformer layers that predict the following word at each position.

57

Encoder-Decoder mdl example: machine translation

However, the decoder layers also attend to the output of the encoder. Consequently, each French output word is conditioned on the previous output words and the entire English sentence it is translating.

This is achieved by modifying the transformer layers in the decoder, that now use a version of self-attention known as encoder-decoder attention or cross-attention where the queries are computed from the decoder embeddings and the keys and values from the encoder embeddings.

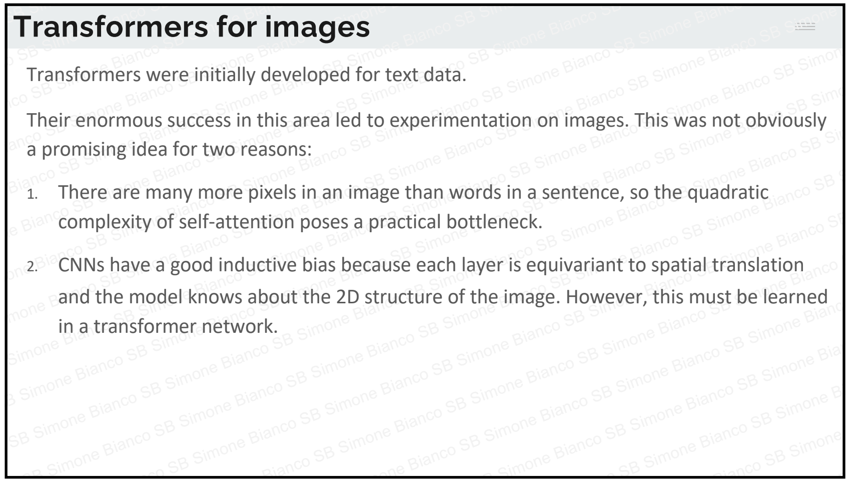

Cross-attention
N_d
D
Decoder Input, $\mathbf{X}_d$
Queries, $\mathbf{Q} = \boldsymbol{\beta}_q \mathbf{1}^T + \boldsymbol{\Omega}_q \mathbf{X}_d$
N_e
Encoder Input, $\mathbf{X}_e$
Keys, $\mathbf{K} = \boldsymbol{\beta}_k \mathbf{1}^T + \boldsymbol{\Omega}_k \mathbf{X}_e$
Values, $\mathbf{V} = \boldsymbol{\beta}_v \mathbf{1}^T + \boldsymbol{\Omega}_v \mathbf{X}_e$
Attention, Softmax $[\mathbf{K}^T\mathbf{Q}]$
Output, $\mathbf{V} \cdot$ Softmax $[\mathbf{K}^T\mathbf{Q}]$

58

AA 2022/2023

Transformers for images

59



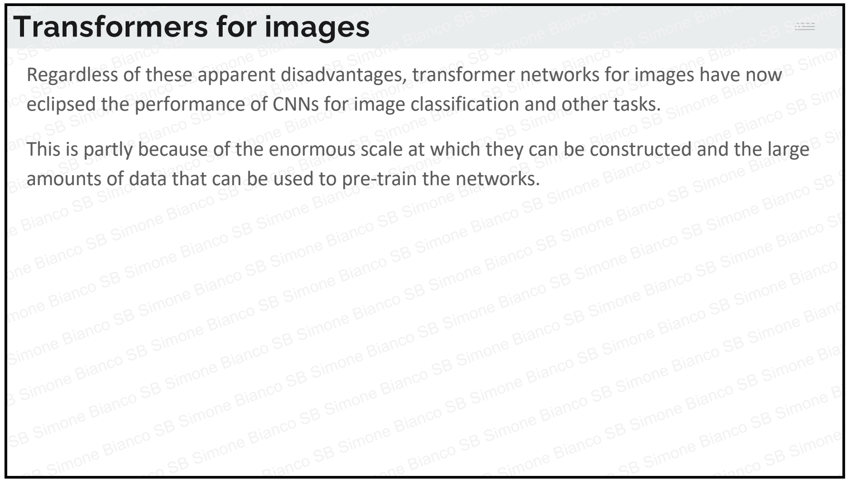
Transformers for images

Transformers were initially developed for text data.

Their enormous success in this area led to experimentation on images. This was not obviously a promising idea for two reasons:

1. There are many more pixels in an image than words in a sentence, so the quadratic complexity of self-attention poses a practical bottleneck.
2. CNNs have a good inductive bias because each layer is equivariant to spatial translation and the model knows about the 2D structure of the image. However, this must be learned in a transformer network.

60

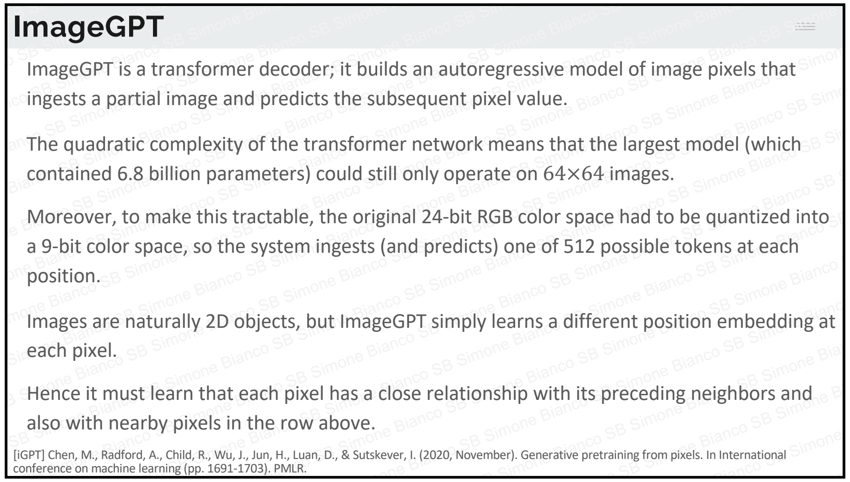


Transformers for images

Regardless of these apparent disadvantages, transformer networks for images have now eclipsed the performance of CNNs for image classification and other tasks.

This is partly because of the enormous scale at which they can be constructed and the large amounts of data that can be used to pre-train the networks.

61



ImageGPT

ImageGPT is a transformer decoder; it builds an autoregressive model of image pixels that ingests a partial image and predicts the subsequent pixel value.

The quadratic complexity of the transformer network means that the largest model (which contained 6.8 billion parameters) could still only operate on 64×64 images.

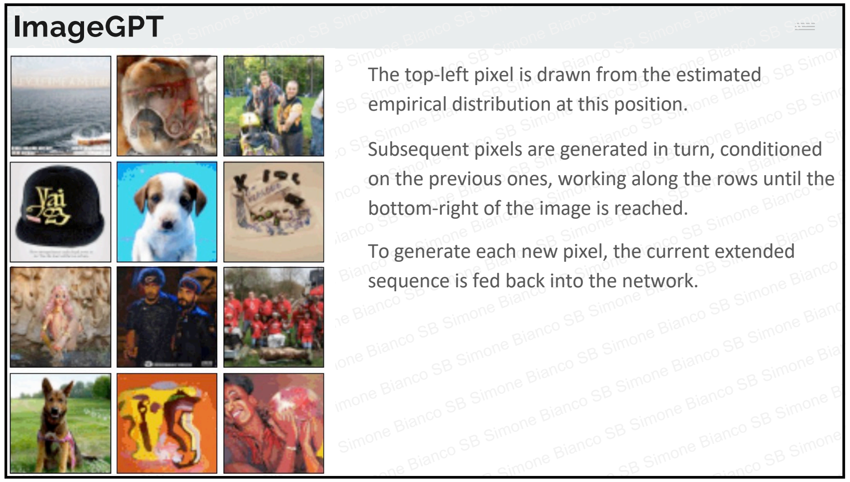
Moreover, to make this tractable, the original 24-bit RGB color space had to be quantized into a 9-bit color space, so the system ingests (and predicts) one of 512 possible tokens at each position.

Images are naturally 2D objects, but ImageGPT simply learns a different position embedding at each pixel.

Hence it must learn that each pixel has a close relationship with its preceding neighbors and also with nearby pixels in the row above.

[iGPT] Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., & Sutskever, I. (2020, November). Generative pretraining from pixels. In International conference on machine learning (pp. 1691-1703). PMLR.

62



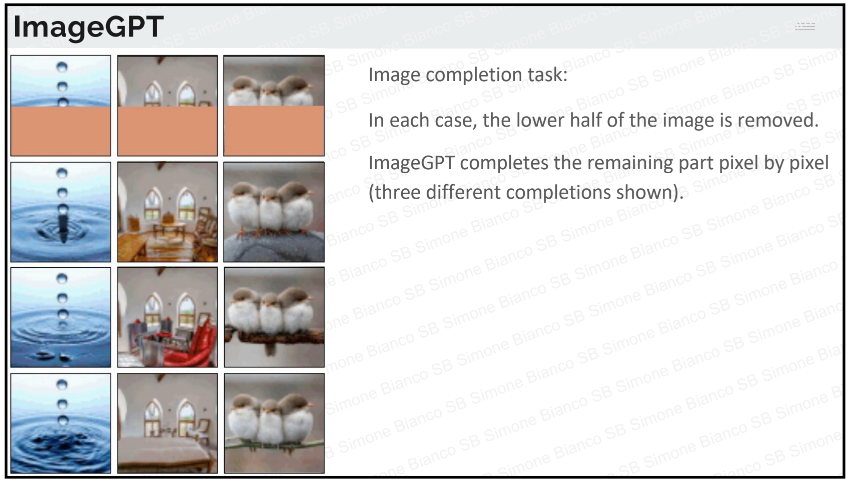
ImageGPT

The top-left pixel is drawn from the estimated empirical distribution at this position.

Subsequent pixels are generated in turn, conditioned on the previous ones, working along the rows until the bottom-right of the image is reached.

To generate each new pixel, the current extended sequence is fed back into the network.

63



ImageGPT

Image completion task:

In each case, the lower half of the image is removed.

ImageGPT completes the remaining part pixel by pixel (three different completions shown).

64

ImageGPT

The internal representation of this decoder was used as a basis for image classification.

The final pixel embeddings are averaged, and a linear layer maps these to activations which are passed through a softmax layer to predict class probabilities.

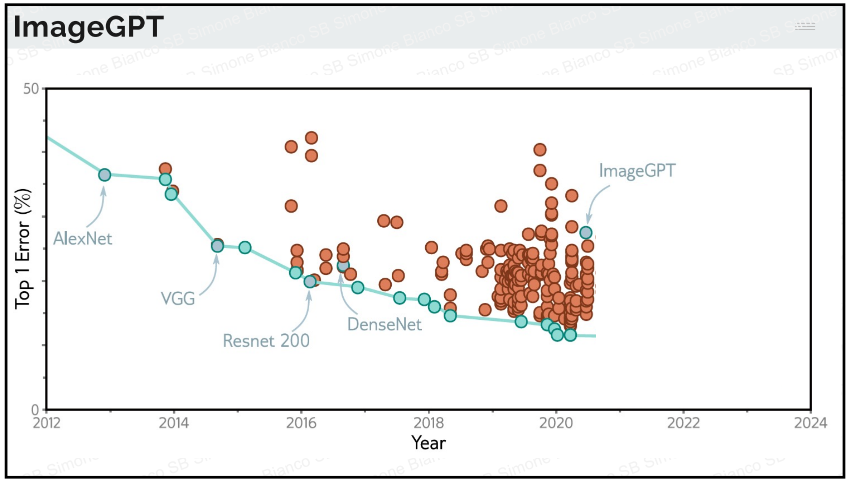
The system is pretrained on a large corpus of web images and then fine-tuned on the ImageNet database resized to 48×48 pixels using a loss function that contains both a cross-entropy term for image classification and a generative loss term for predicting the pixels.

Despite using a large amount of external training data, the system achieved only a 27.4% top-1 error rate on ImageNet.

This was less than convolutional architectures of the time but is still impressive given the small input image size.

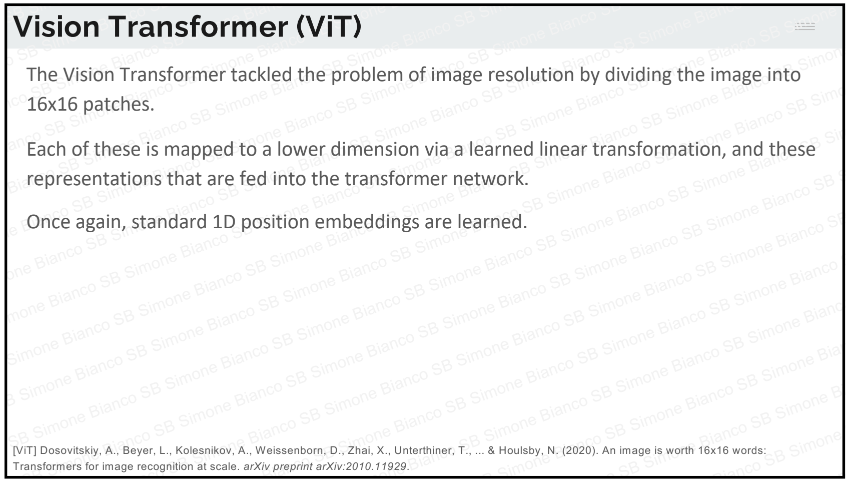
Unsurprisingly, it fails to classify images where the target object is small or thin.

65



ImageGPT

66



Vision Transformer (ViT)

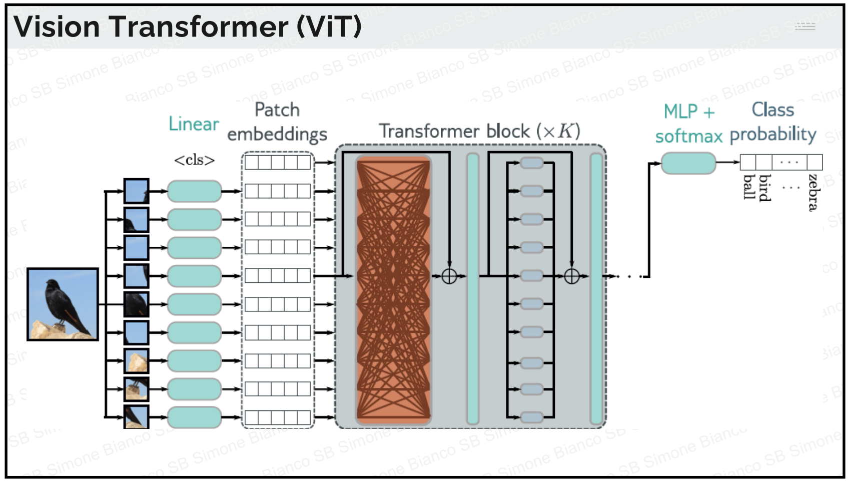
The Vision Transformer tackled the problem of image resolution by dividing the image into 16x16 patches.

Each of these is mapped to a lower dimension via a learned linear transformation, and these representations that are fed into the transformer network.

Once again, standard 1D position embeddings are learned.

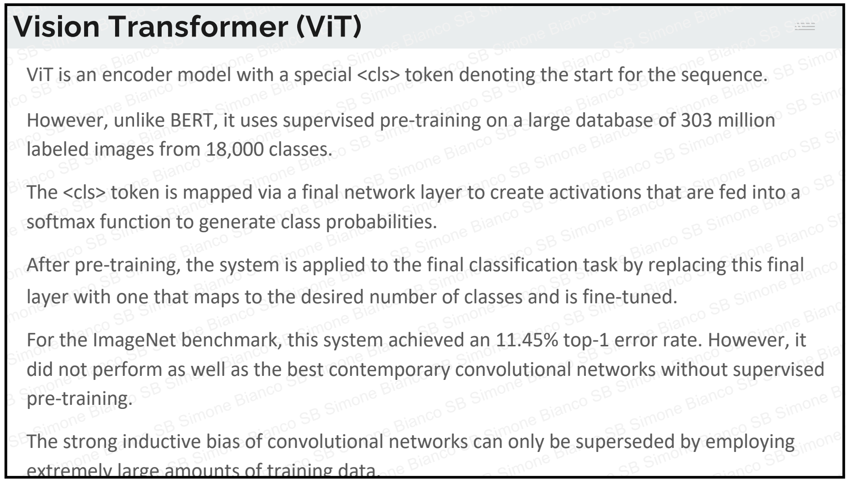
[ViT] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929.*

67



Vision Transformer (ViT)

68



Vision Transformer (ViT)

ViT is an encoder model with a special <cls> token denoting the start for the sequence.

However, unlike BERT, it uses supervised pre-training on a large database of 303 million labeled images from 18,000 classes.

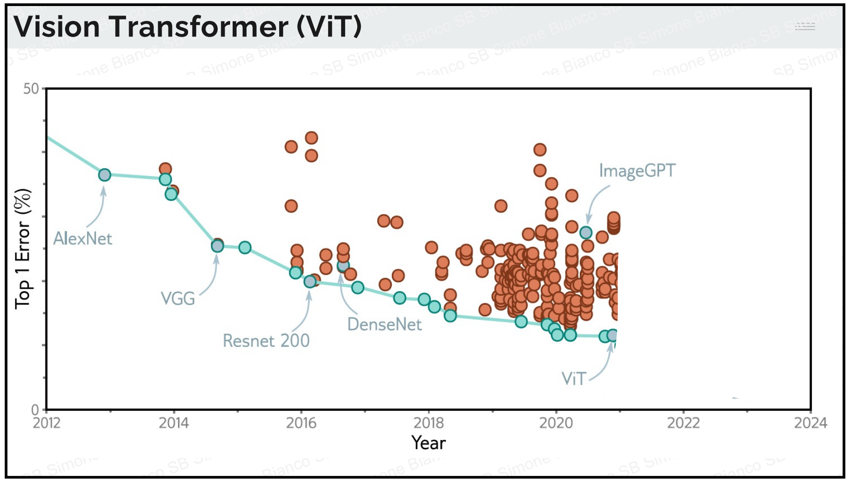
The <cls> token is mapped via a final network layer to create activations that are fed into a softmax function to generate class probabilities.

After pre-training, the system is applied to the final classification task by replacing this final layer with one that maps to the desired number of classes and is fine-tuned.

For the ImageNet benchmark, this system achieved an 11.45% top-1 error rate. However, it did not perform as well as the best contemporary convolutional networks without supervised pre-training.

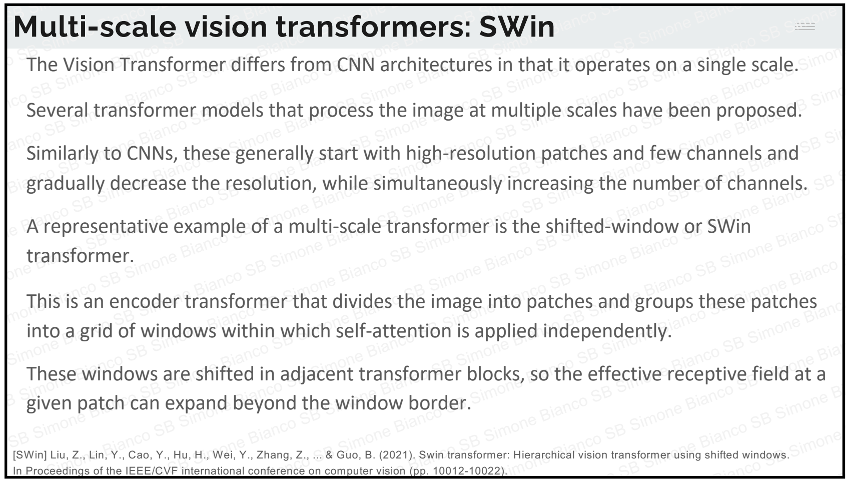
The strong inductive bias of convolutional networks can only be superseded by employing extremely large amounts of training data.

69



Vision Transformer (ViT)

70



Multi-scale vision transformers: SWin

The Vision Transformer differs from CNN architectures in that it operates on a single scale.

Several transformer models that process the image at multiple scales have been proposed.

Similarly to CNNs, these generally start with high-resolution patches and few channels and gradually decrease the resolution, while simultaneously increasing the number of channels.

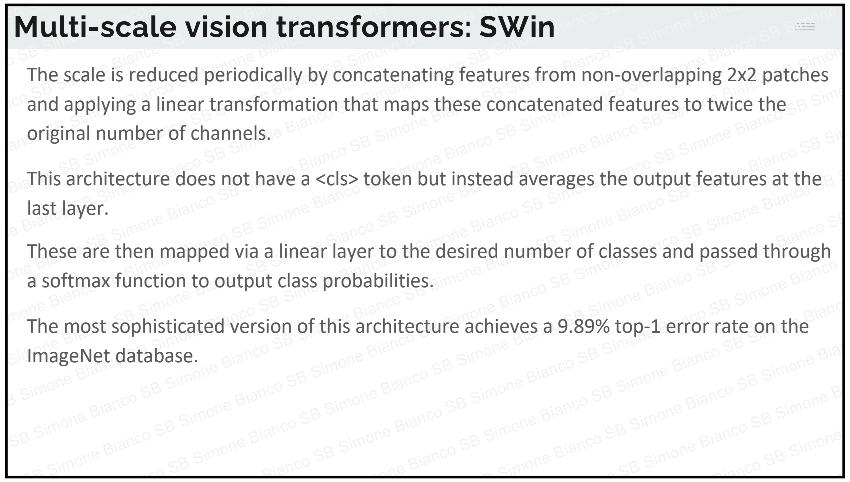
A representative example of a multi-scale transformer is the shifted-window or SWin transformer.

This is an encoder transformer that divides the image into patches and groups these patches into a grid of windows within which self-attention is applied independently.

These windows are shifted in adjacent transformer blocks, so the effective receptive field at a given patch can expand beyond the window border.

[SWin] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 10012-10022).

71



Multi-scale vision transformers: SWin

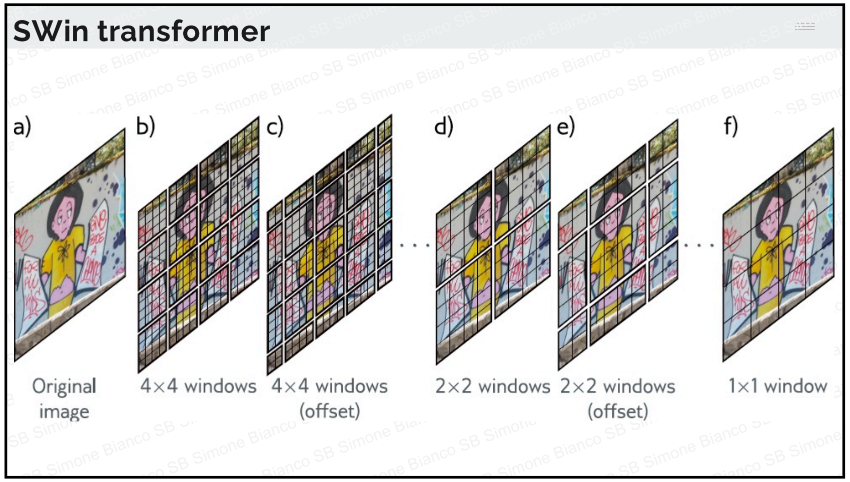
The scale is reduced periodically by concatenating features from non-overlapping 2x2 patches and applying a linear transformation that maps these concatenated features to twice the original number of channels.

This architecture does not have a <cls> token but instead averages the output features at the last layer.

These are then mapped via a linear layer to the desired number of classes and passed through a softmax function to output class probabilities.

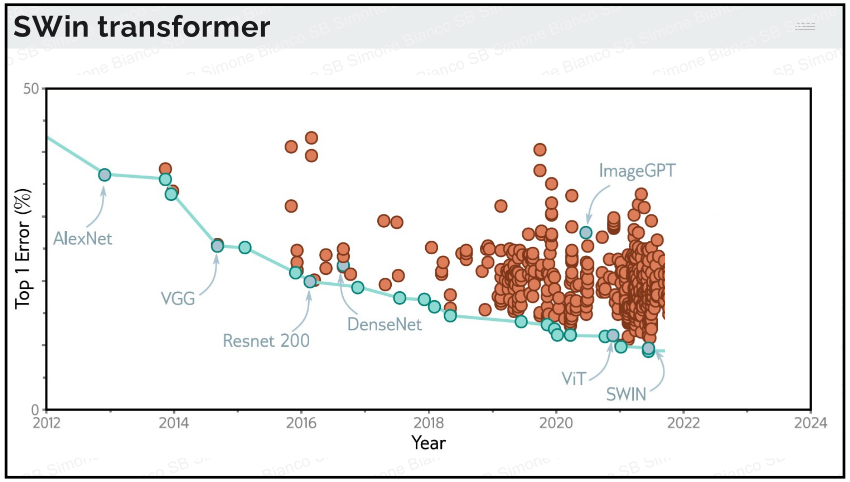
The most sophisticated version of this architecture achieves a 9.89% top-1 error rate on the ImageNet database.

72



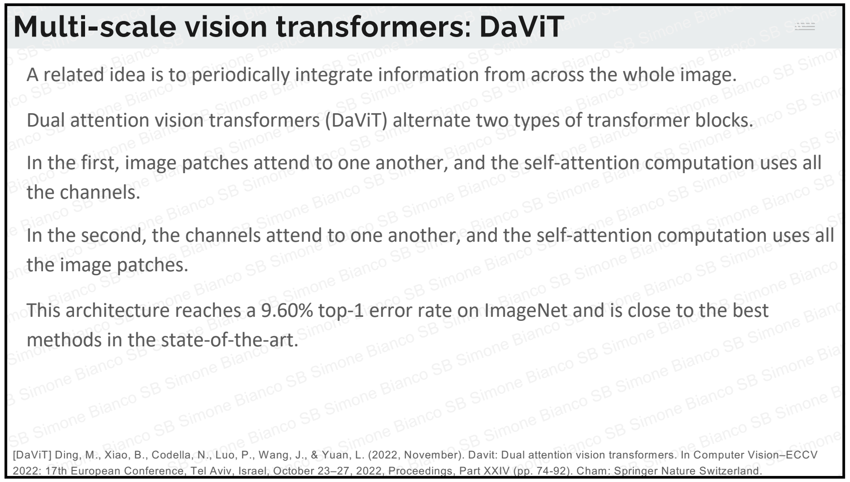
SWin transformer

73



SWin transformer

74



Multi-scale vision transformers: DaViT

A related idea is to periodically integrate information from across the whole image.

Dual attention vision transformers (DaViT) alternate two types of transformer blocks.

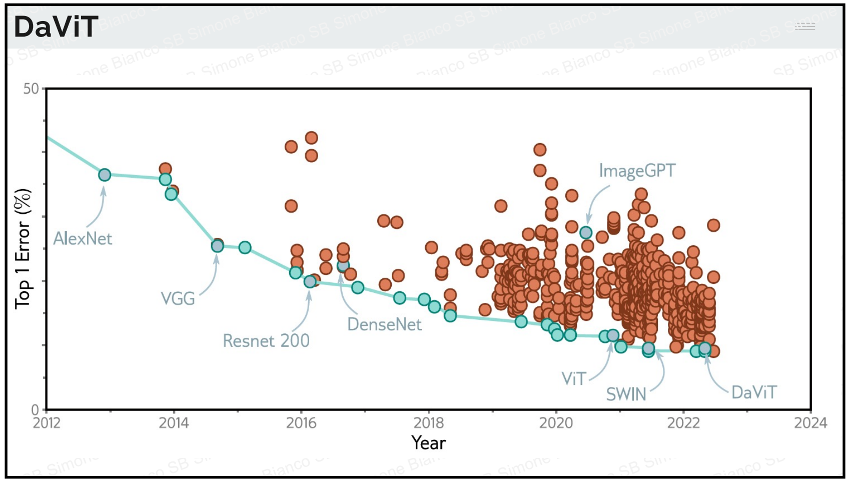
In the first, image patches attend to one another, and the self-attention computation uses all the channels.

In the second, the channels attend to one another, and the self-attention computation uses all the image patches.

This architecture reaches a 9.60% top-1 error rate on ImageNet and is close to the best methods in the state-of-the-art.

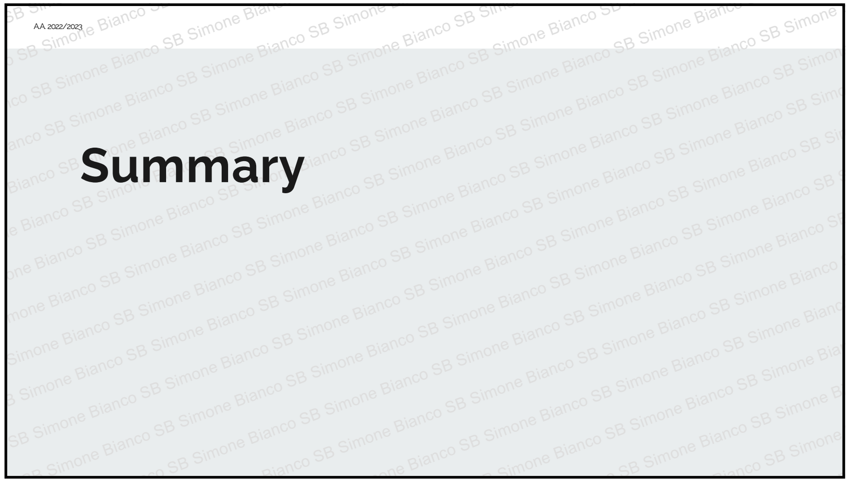
[DaViT] Ding, M., Xiao, B., Codella, N., Luo, P., Wang, J., & Yuan, L. (2022, November). Davit: Dual attention vision transformers. In Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXIV (pp. 74-92). Cham: Springer Nature Switzerland.

75



DaViT

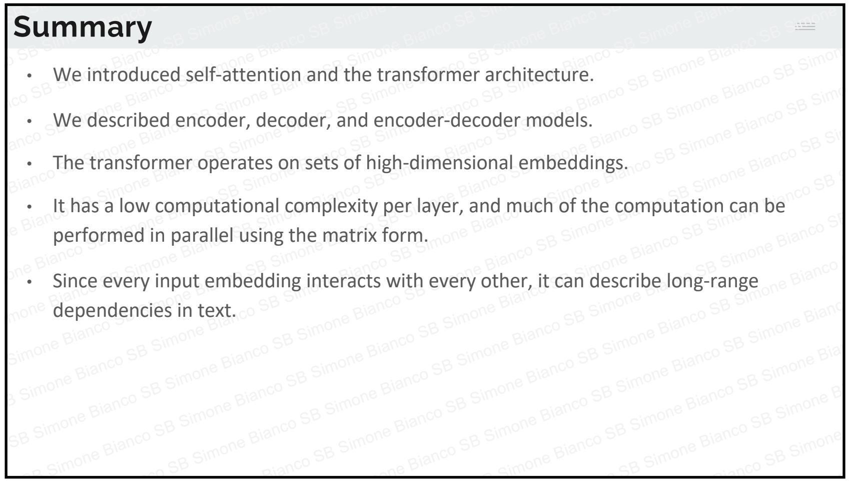
76



AA 2022/2023

Summary

77



Summary

- We introduced self-attention and the transformer architecture.
- We described encoder, decoder, and encoder-decoder models.
- The transformer operates on sets of high-dimensional embeddings.
- It has a low computational complexity per layer, and much of the computation can be performed in parallel using the matrix form.
- Since every input embedding interacts with every other, it can describe long-range dependencies in text.

78