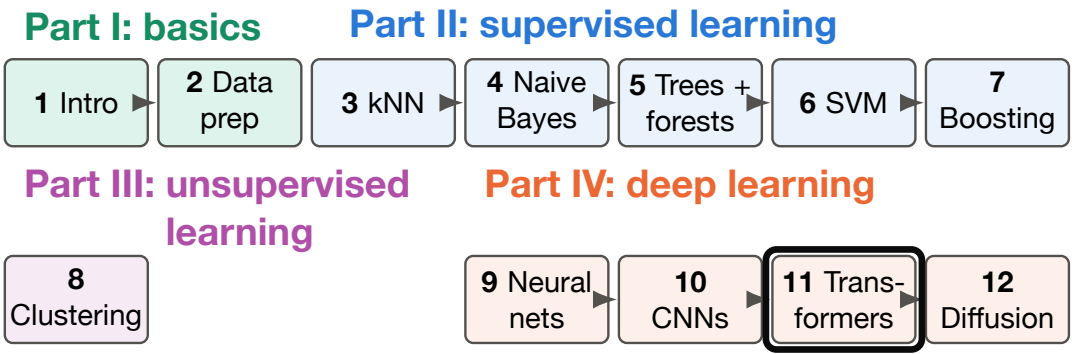
Transformers

229352 Lecture 11: transformers

Course map


Part I: basics
Part II: supervised learning
1 Intro
2 Data prep
3 kNN
4 Naive Bayes
5 Trees + forests
6 SVM
7 Boosting
Part III: unsupervised learning
Part IV: deep learning
8 Clustering
9 Neural nets
10 CNNs
11 Trans-formers
12 Diffusion

Attention is all you need

Attention is all you need

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones,
A. N. Gomez, Ł. Kaiser and I. Polosukhin

NIPS 2017

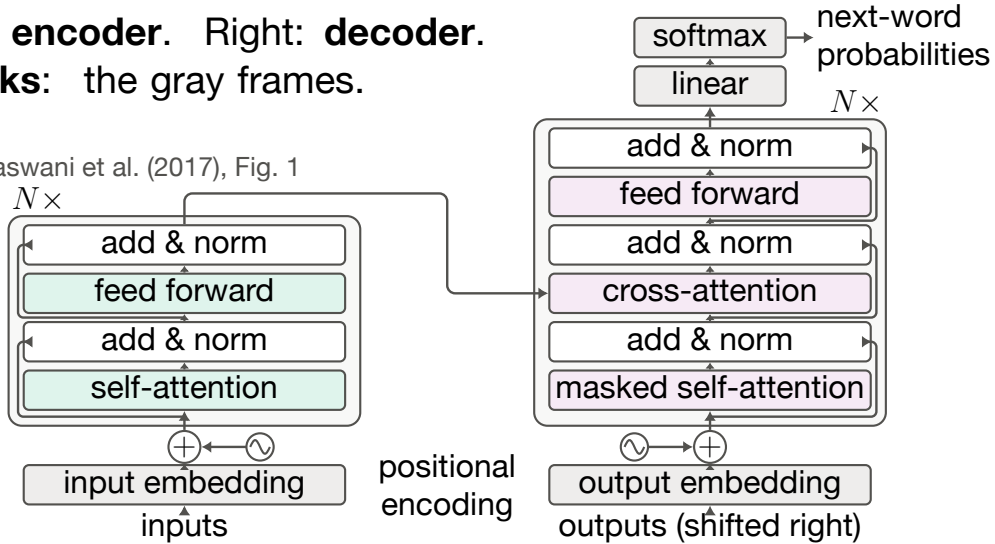
It introduced the **transformer**, a network for translation.

GPT, BERT and ChatGPT (end of this lecture)
use parts of the transformer.

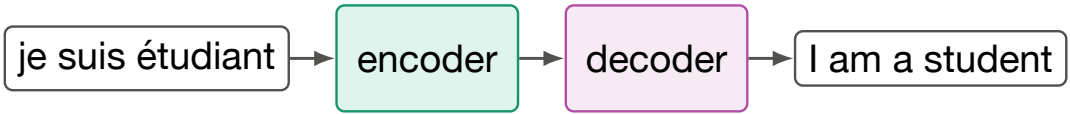
The transformer

Left: **encoder**. Right: **decoder**.
Blocks: the gray frames.

after Vaswani et al. (2017), Fig. 1



The big picture

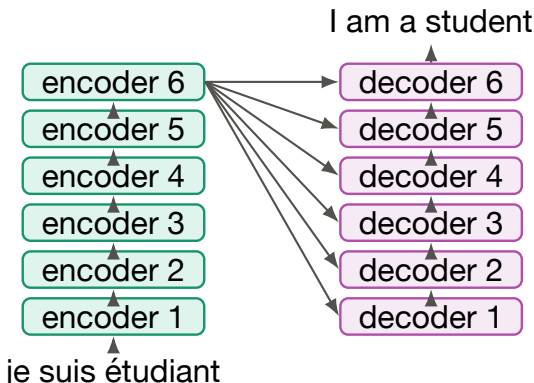


In our example, we translate French “je suis étudiant” into English “I am a student”.

The encoder turns each French word into a vector, using the whole sentence.

The decoder uses these vectors to write the English sentence, one word at a time.

A smaller picture: stacks of blocks



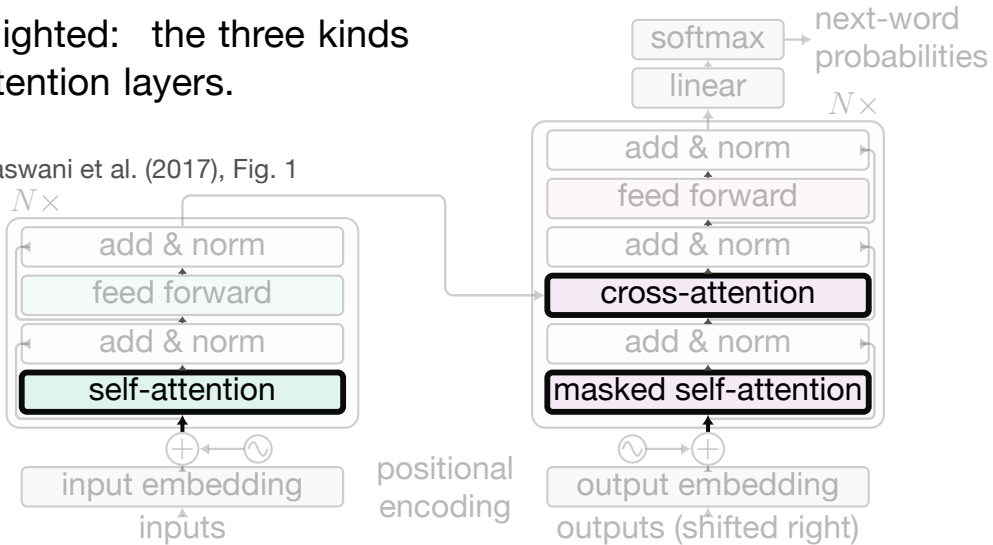
Blocks of a stack have the same structure, but their own parameters.

The output of the last encoder block goes to every decoder block.

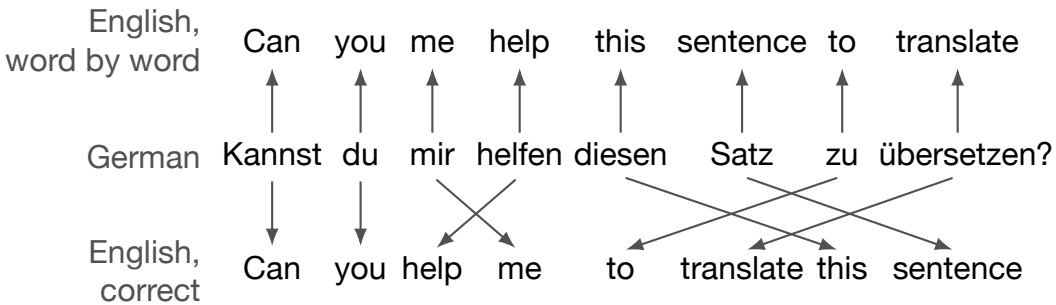
Attention layers

Highlighted: the three kinds of attention layers.

after Vaswani et al. (2017), Fig. 1

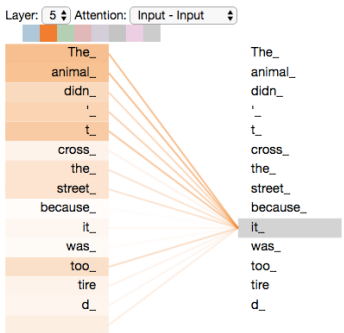


Which words matter for a word?



To translate a word, the network must use the words that belong with it, which may be far away.

Attention in a trained transformer



Self-attention: the output for a word is a weighted average over all words of its sentence.

One set of weights (an attention **head**) in the top encoder block: the weights of “it” (right) on all words (left). A darker, larger line is a larger weight.

Largest weights: “The” and “animal”, which “it” refers to.

J. Alammar, *The Illustrated Transformer*, 2018, CC BY-NC-SA 4.0; Google’s Tensor2Tensor `hello_t2t`

The running example

Sentence: “I am a student”, $n = 4$ words; here each word is one **token** (a token is a word or a piece of a word, such as “tire” + “d”).

Word i: **embedding** $x^{(i)}$, a row of $d_{\text{model}} = 4$ numbers (superscript: the word, not a layer).

X: the $n \times d_{\text{model}}$ matrix with rows $x^{(1)}, \ldots, x^{(4)}$; random here, learned in training.

$$X = \begin{array}{r} \text{I} \\ \text{am} \\ \text{a} \\ \text{student} \end{array} \begin{pmatrix} 0.1 & -0.1 & 0.6 & 0.1 \\ -0.5 & 0.4 & 1.3 & 0.9 \\ -0.7 & -1.3 & -0.6 & 0.0 \\ -2.3 & -0.2 & -1.2 & -0.7 \end{pmatrix}$$

Step 1: three learned matrices

Each word gets a **query** vector, a **key** vector and a **value** vector ($d_k = 3$ entries):

$$q^{(i)} = x^{(i)} W^Q, \qquad k^{(i)} = x^{(i)} W^K, \qquad v^{(i)} = x^{(i)} W^V$$

$$W^Q = \begin{pmatrix} -0.3 & -0.2 & 0.2 \\ 0.5 & -0.1 & 0.7 \\ -0.3 & 0.2 & 0.5 \\ 0.0 & -0.4 & -0.5 \end{pmatrix} \qquad W^K = \begin{pmatrix} -0.2 & 0.1 & -0.5 \\ -0.1 & -0.1 & 0.3 \\ 0.1 & 0.2 & -0.3 \\ -0.1 & 0.4 & 0.7 \end{pmatrix} \qquad W^V = \begin{pmatrix} -0.6 & 0.8 & 0.7 \\ 0.4 & 0.1 & -0.2 \\ 0.7 & 1.0 & 0.9 \\ 0.7 & 0.2 & -0.6 \end{pmatrix}$$

Each matrix is $d_{\text{model}} \times d_k = 4 \times 3$.
Learned in training, random here.

Step 1: queries, keys and values

Rows of $Q = XW^Q$, $K = XW^K$, $V = XW^V$: $q^{(i)}$, $k^{(i)}$, $v^{(i)}$, rounded to one decimal.

$$
\begin{array}{r} \\ \text{I} \\ \text{am} \\ \text{a} \\ \text{student} \end{array}
\begin{array}{c} Q \\ \begin{pmatrix} -0.3 & 0.1 & 0.2 \\ 0.0 & 0.0 & 0.4 \\ -0.3 & 0.2 & -1.4 \\ 1.0 & 0.5 & -0.9 \end{pmatrix} \end{array}
\quad
\begin{array}{c} K \\ \begin{pmatrix} 0.0 & 0.2 & -0.2 \\ 0.1 & 0.5 & 0.6 \\ 0.2 & -0.1 & 0.1 \\ 0.4 & -0.7 & 1.0 \end{pmatrix} \end{array}
\quad
\begin{array}{c} V \\ \begin{pmatrix} 0.4 & 0.7 & 0.6 \\ 2.0 & 1.1 & 0.2 \\ -0.5 & -1.3 & -0.8 \\ 0.0 & -3.2 & -2.2 \end{pmatrix} \end{array}
$$

Query–key dot products give scores, then weights.
The values are averaged with these weights.

Step 2: scores for the word “I”

Query of “I”: $q^{(1)} = (-0.3,\ 0.1,\ 0.2)$. Score of word j:

$$s_{1j} = q^{(1)} \cdot k^{(j)} = q_1^{(1)} k_1^{(j)} + q_2^{(1)} k_2^{(j)} + q_3^{(1)} k_3^{(j)}$$

word j	**key** $k^{(j)}$	**score** s_{1j}
I	$(0.0,\ 0.2,\ -0.2)$	-0.02
am	$(0.1,\ 0.5,\ 0.6)$	0.14
a	$(0.2,\ -0.1,\ 0.1)$	-0.05
student	$(0.4,\ -0.7,\ 1.0)$	0.01

$q_m^{(1)}$, $k_m^{(j)}$: entry m. The keys are the rows of K.

Step 3: divide by $\sqrt{d_k}$, then softmax

word j	s_{1j}	$s_{1j}/\sqrt{3}$	a_{1j}
I	-0.02	-0.01	0.24
am	0.14	0.08	0.27
a	-0.05	-0.03	0.24
student	0.01	0.01	0.25

Divide by $\sqrt{d_k} = \sqrt{3} \approx 1.73$.

Softmax (Lecture 9) over the four words: the weights are positive and sum to 1.

$$a_{1j} = \frac{e^{s_{1j}/\sqrt{d_k}}}{e^{s_{11}/\sqrt{d_k}} + \cdots + e^{s_{14}/\sqrt{d_k}}}$$

Here the scores of "I" are small (-0.05 to 0.14), so its four weights are close to $1/4$.

Step 4: weighted sum of the values

word j	**weight** a_{1j}	**value** $v^{(j)}$	$a_{1j}\, v^{(j)}$
I	0.24	$(0.4,\ 0.7,\ 0.6)$	$(0.096,\ 0.168,\ 0.144)$
am	0.27	$(2.0,\ 1.1,\ 0.2)$	$(0.540,\ 0.297,\ 0.054)$
a	0.24	$(-0.5,\ -1.3,\ -0.8)$	$(-0.120,\ -0.312,\ -0.192)$
student	0.25	$(0.0,\ -3.2,\ -2.2)$	$(0.000,\ -0.800,\ -0.550)$

Output for “I”: the sum of the last column,

$$z^{(1)} = \sum_j a_{1j}\, v^{(j)} = (0.516,\ -0.647,\ -0.544)$$

rounded: $(0.52,\ -0.65,\ -0.54)$.

Each value $v^{(j)}$ enters $z^{(1)}$ times its weight a_{1j}; a weight near 0 removes word j.

All four words at once

Steps 2 to 4 for every word, one row per word:

$$A = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right), \qquad Z = AV$$

$$A = \begin{array}{r} \text{I} \\ \text{am} \\ \text{a} \\ \text{student} \end{array} \begin{pmatrix} 0.24 & 0.27 & 0.24 & 0.25 \\ 0.22 & 0.26 & 0.23 & 0.29 \\ 0.39 & 0.21 & 0.28 & 0.12 \\ 0.32 & 0.24 & 0.28 & 0.16 \end{pmatrix} \qquad Z = \begin{pmatrix} 0.52 & -0.65 & -0.54 \\ 0.49 & -0.79 & -0.64 \\ 0.44 & -0.24 & -0.21 \\ 0.47 & -0.39 & -0.34 \end{pmatrix}$$

The softmax is taken row by row: every row of A sums to 1.

Row 1 of A and of Z is the result of steps 3 and 4 (Z rounded).

Self-attention in one equation

$$Z = \text{Attention}(Q, K, V) = \underbrace{\text{softmax}\left(\frac{\overbrace{QK^\top}^{n \times n \text{ scaled scores}}}{\sqrt{d_k}}\right)}_{A\text{: weights, rows sum to 1}} V$$

Q, K, V, Z: $n \times d_k$, one row per word; $z^{(i)} = \sum_j a_{ij} v^{(j)}$.

a_{ij}: the weight of word j in the output of word i.

Vaswani et al. (2017), Section 3.2.1

Summary: self-attention

- Each word i gets $q^{(i)} = x^{(i)}W^Q$, $k^{(i)} = x^{(i)}W^K$ and $v^{(i)} = x^{(i)}W^V$; W^Q, W^K, W^V are learned.
- Scores $s_{ij} = q^{(i)} \cdot k^{(j)}$, divided by $\sqrt{d_k}$; the softmax over j gives weights a_{ij} that sum to 1.
- Output $z^{(i)} = \sum_j a_{ij}\, v^{(j)}$: a weighted average of the values of all words.
- All words at once:
  $\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(QK^\top/\sqrt{d_k}\right)V.$

Checkpoint 1

Why divide the scores by $\sqrt{d_k}$?

Hint: independent entries of q and k, mean 0, variance 1.

Checkpoint 1

Why divide the scores by $\sqrt{d_k}$?

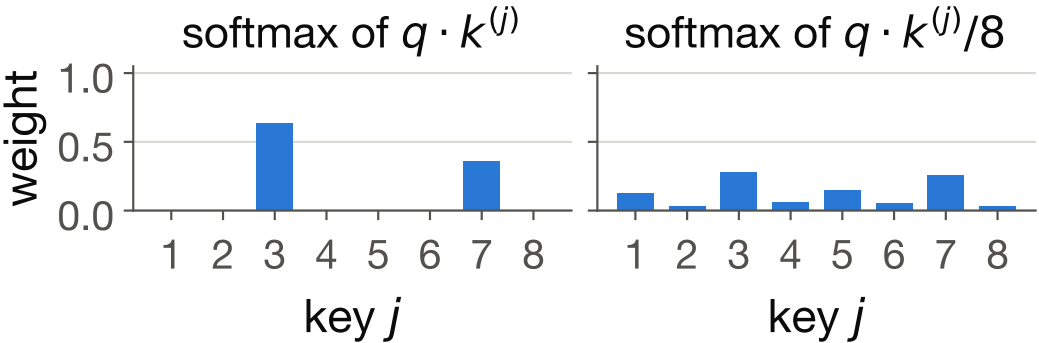
Hint: independent entries of q and k, mean 0, variance 1.

$q \cdot k = q_1 k_1 + \cdots + q_{d_k} k_{d_k}$ has mean 0 and variance d_k (standard deviation $\sqrt{d_k}$), so the scores are large in size when d_k is large.

The softmax then puts almost all the weight on the few largest scores, and the other weights are almost 0. A weight near 0 hardly changes when its score changes (tiny gradients).

Dividing by $\sqrt{d_k}$ brings the variance back to 1.

Scores with and without the division



$d_k = 64$ ($\sqrt{d_k} = 8$); 8 keys $k^{(j)}$; entries: mean 0, variance 1

without the division, 6 of the 8 weights are below 0.01

One word, two meanings

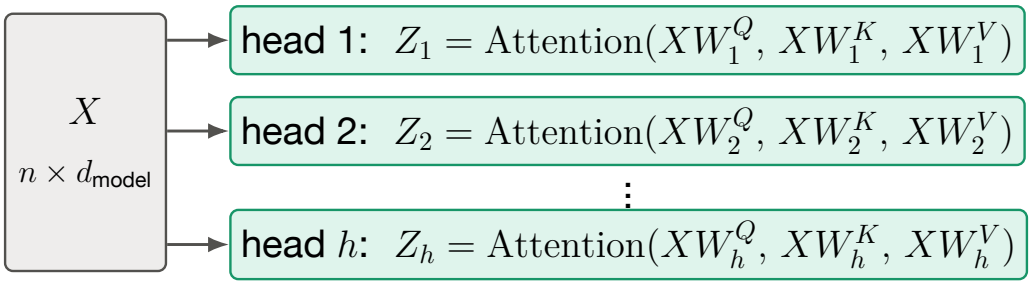
"She turned on the **light.**"

"The bag was very **light.**"

"light" belongs with "turned on" in the first sentence (a lamp) and with "bag" in the second (not heavy).

One set of weights a_{ij} describes one kind of relation.
Multi-head attention computes several sets (heads) at once.

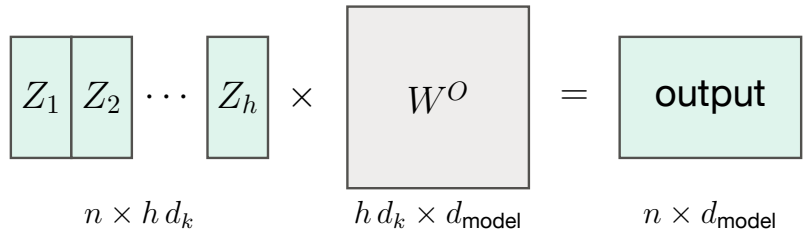
Multi-head attention: h heads



Each head has its own W^Q, W^K, W^V (subscript: the head) and computes self-attention as before.

Each W of a head is $d_{\text{model}} \times d_k$, so each Z is $n \times d_k$.

Combining the heads



Place the h outputs side by side, then multiply by a learned matrix W^O:

$$\text{MultiHead}(X) = \left[\, Z_1 \; Z_2 \; \cdots \; Z_h \,\right] W^O$$

The output has the shape of X.

Two heads on the running example

	I	am	a	student
I	0.23	0.29	0.23	0.25
am	0.22	0.14	0.36	0.28
a	0.32	0.42	0.12	0.14
student	0.36	0.30	0.17	0.17

head 1: weights A_1

	I	am	a	student
I	0.20	0.15	0.30	0.35
am	0.11	0.06	0.31	0.52
a	0.03	0.01	0.25	0.71
student	0.06	0.04	0.21	0.69

head 2: weights A_2

subscript: the head; $d_k = d_{\text{model}}/h = 2$; random W^Q, W^K

largest weight of “I”: “am” in head 1, “student” in head 2

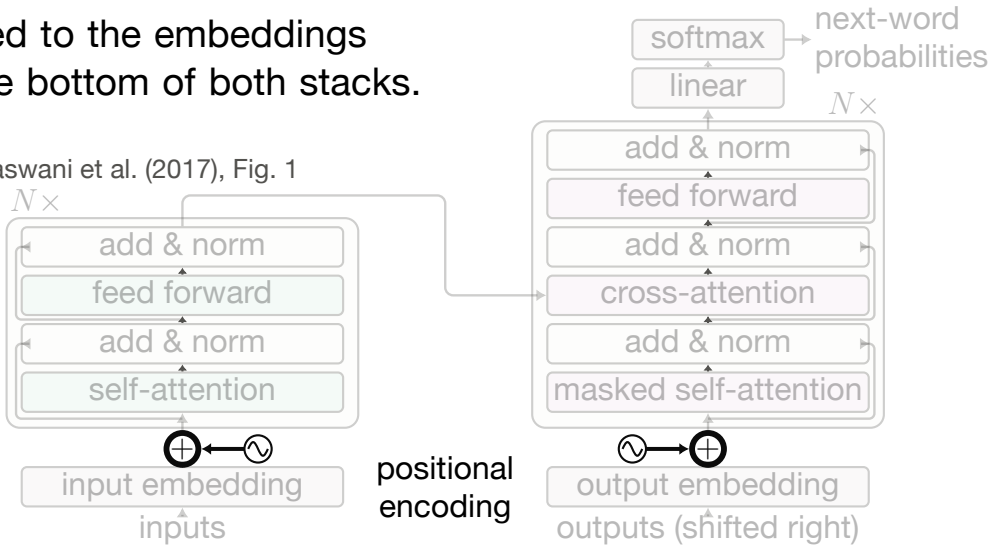
Summary: multi-head attention

- A word can relate to other words in several ways; one set of weights a_{ij} gives one way.
- h heads, each with its own W^Q, W^K, W^V of size $d_{\text{model}} \times d_k$, each computing self-attention.
- The h outputs, placed side by side and multiplied by W^O, give an $n \times d_{\text{model}}$ matrix, the shape of X.

Positional encoding

Added to the embeddings
at the bottom of both stacks.

after Vaswani et al. (2017), Fig. 1



Attention does not use word order

Swap “I” and “student”: “student am a I”.

Self-attention with the same W^Q, W^K, W^V gives the same rows of Z, in the new order.

Output for “I”: $(0.52,\ -0.65,\ -0.54)$ in “I am a student”, and $(0.52,\ -0.65,\ -0.54)$ in “student am a I”.

The scores, the weights and the sum $\sum_j a_{ij} v^{(j)}$ do not use the position of word j.

So the output for each word is the same in both sentences: self-attention alone does not use the word order.

Adding the positional encoding

$x^{(1)}$		PE_0	$x^{(2)}$		PE_1	$x^{(3)}$		PE_2	$x^{(4)}$		PE_3
0.1		0.00	−0.5		0.84	−0.7		0.91	−2.3		0.14
−0.1	+	1.00	0.4	+	0.54	−1.3	+	−0.42	−0.2	+	−0.99
0.6		0.00	1.3		0.01	−0.6		0.02	−1.2		0.03
0.1		1.00	0.9		1.00	0.0		1.00	−0.7		1.00
I ($\mathrm{pos} = 0$)			am ($\mathrm{pos} = 1$)			a ($\mathrm{pos} = 2$)			student ($\mathrm{pos} = 3$)		

$\mathrm{PE}_{\mathrm{pos}}$: the **positional encoding**, a fixed vector for position pos (counted from 0). Word i is at position $i - 1$, so its input is $x^{(i)} + \mathrm{PE}_{i-1}$.

Output for “I” (inputs rounded to one decimal): $(0.85,\ -0.39,\ -1.03)$ when “I” is first, $(0.68,\ 0.21,\ -0.53)$ when it is last.

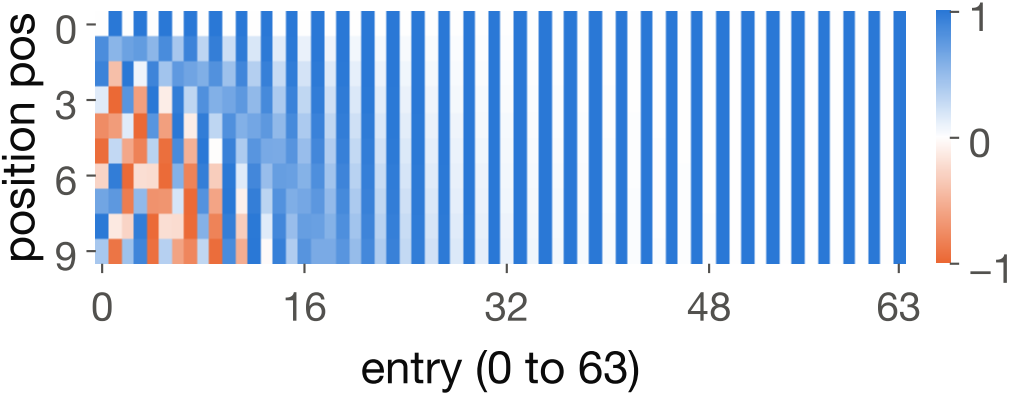
Sinusoidal positional encoding

$$\mathrm{PE}_{(\mathrm{pos},\,2m)} = \sin\big(\,\mathrm{pos}\,/\theta_m\big), \qquad \mathrm{PE}_{(\mathrm{pos},\,2m+1)} = \cos\big(\,\mathrm{pos}\,/\theta_m\big)$$

$$\theta_m = 10000^{2m/d_{\text{model}}}$$

- Subscripts: the position $\mathrm{pos} = 0, 1, 2, \ldots$ and the entry, both counted from 0.
- Entries $2m$ and $2m+1$ (the pair $m = 0, \ldots, d_{\text{model}}/2 - 1$) repeat every $2\pi\,\theta_m$ positions: from 2π to almost $10000 \cdot 2\pi$.
- $d_{\text{model}} = 4$: $\theta_0 = 1$, $\theta_1 = 100$, so $\mathrm{PE}_1 = (\sin 1, \cos 1, \sin 0.01, \cos 0.01)$.

Positional encodings of 10 positions



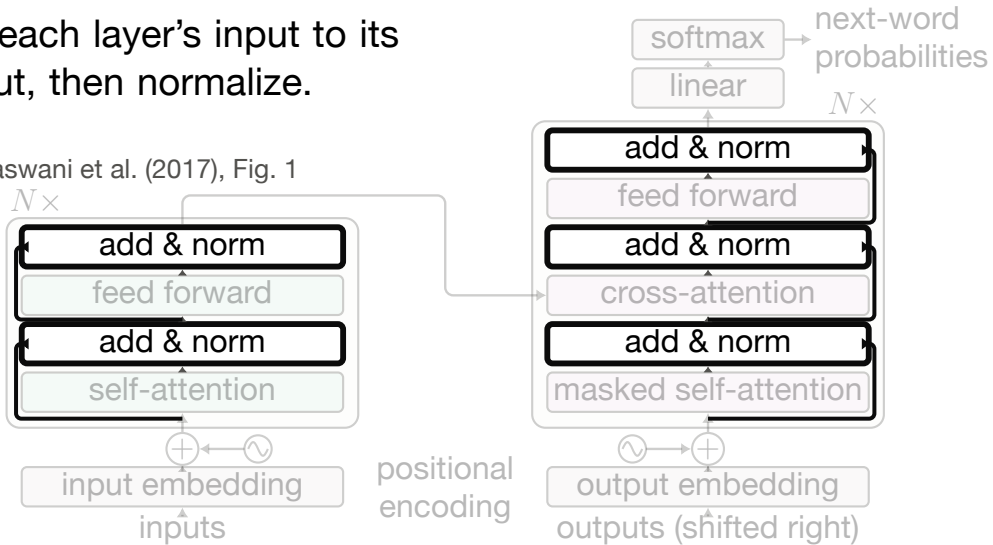
$d_{\text{model}} = 64$, positions 0 to 9; blue: $+1$, orange: -1

entries on the left change fast; on the right, slowly

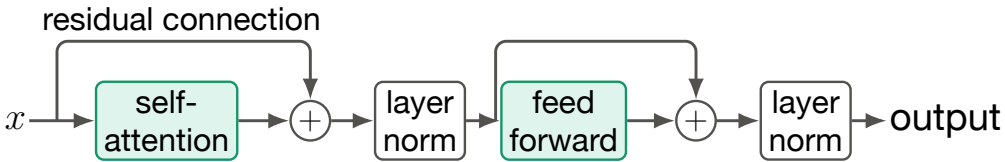
Add & norm

Add each layer's input to its output, then normalize.

after Vaswani et al. (2017), Fig. 1



Residual connection and layer norm

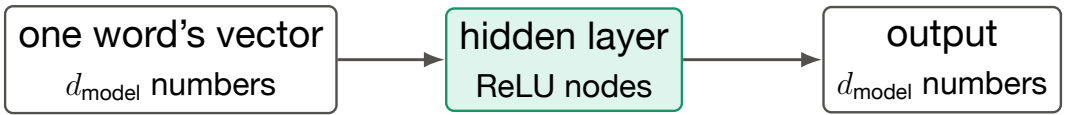


One encoder block, for row x of its input (one word); self-attention also uses the other words' rows.

Residual connection: the input skips the layer and is added to its output: $x + \text{Sublayer}(x)$, where $\text{Sublayer}(x)$ is the output of the attention or feed-forward layer.

Then layer normalization: $\text{LayerNorm}\left(x + \text{Sublayer}(x)\right)$.

The feed-forward layer



Every block has, after its attention layer (two in a decoder block), a network with one hidden layer (Lecture 9).

It is applied to each word's vector separately, with the same parameters for every word.

Review: batch normalization

#	f_1	f_2	f_3	f_4
1	8	1	2	3
2	2	8	8	6
3	1	1	3	4
mean	3.67	3.33	4.33	4.33
sd	3.09	3.30	2.62	1.25

$\rightarrow$

#	f_1	f_2	f_3	f_4
1	1.40	-0.71	-0.89	-1.07
2	-0.54	1.41	1.40	1.34
3	-0.86	-0.71	-0.51	-0.27

Made up: 3 data points (rows), 4 features $f_1, \ldots, f_4$ (columns); sd: standard deviation.

Batch normalization standardizes each feature over the minibatch (Lecture 10: each channel, over positions too).

Right: the standardized values, (value $-$ column mean) / column sd.

Layer normalization

#	f_1	f_2	f_3	f_4	mean	sd
1	8	1	2	3	3.50	2.69
2	2	8	8	6	6.00	2.45
3	1	1	3	4	2.25	1.30

$\rightarrow$

#	f_1	f_2	f_3	f_4
1	1.67	-0.93	-0.56	-0.19
2	-1.63	0.82	0.82	0.00
3	-0.96	-0.96	0.58	1.35

Layer normalization standardizes each data point (row) over its own features. In a transformer, a row is one word's vector.

Right: the standardized values, (value $-$ row mean) / row sd. The other data points are not used.

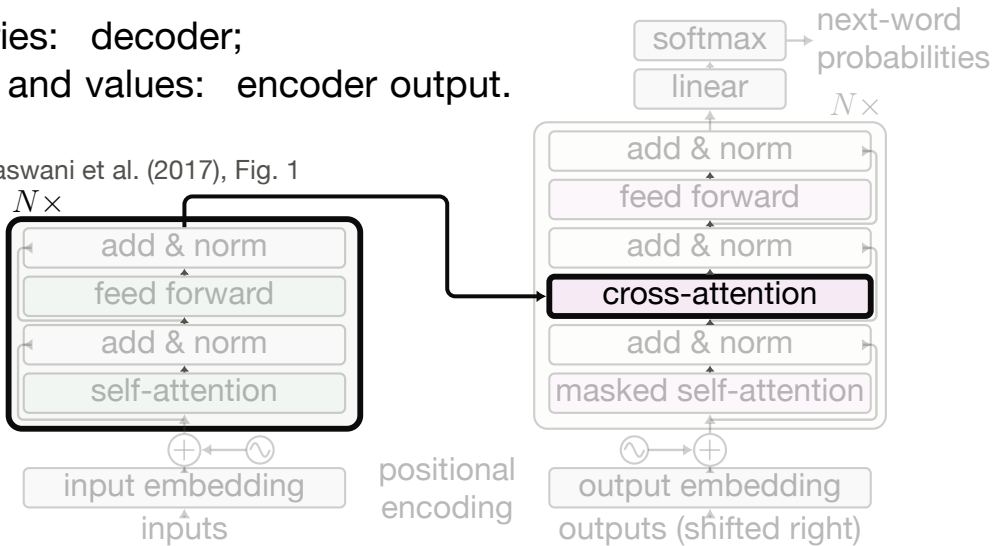
Summary: the encoder

- Input of the first block: for each word, its embedding plus the positional encoding of its position.
- Each of the N encoder blocks: multi-head self-attention, then a feed-forward network, each followed by add & norm: $\mathrm{LayerNorm}\left(x + \mathrm{Sublayer}(x)\right)$.
- Layer normalization standardizes each word's vector; batch normalization standardizes each feature over the minibatch.
- Every block preserves the shape $n \times d_{\text{model}}$, so the blocks can be stacked.

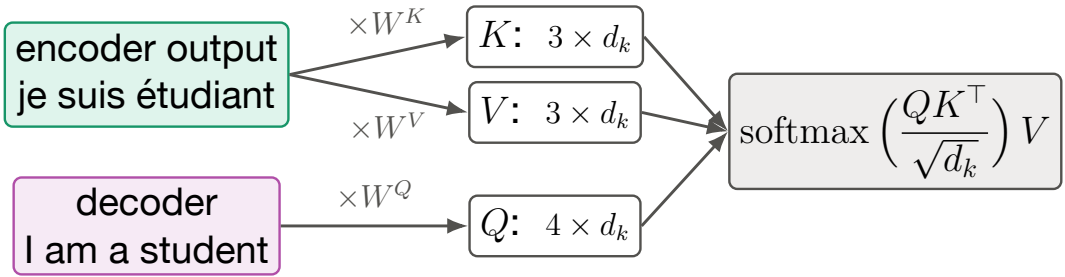
The decoder: cross-attention

Queries: decoder;
keys and values: encoder output.

after Vaswani et al. (2017), Fig. 1



Cross-attention



The formula is that of self-attention, but Q comes from the decoder, and K and V from the encoder.

The weights form a 4×3 matrix: one row per English word, one column per French word.

Cross-attention weights

	je	suis	étudiant
I	0.31	0.32	0.37
am	0.16	0.13	0.71
a	0.40	0.34	0.26
student	0.34	0.37	0.29

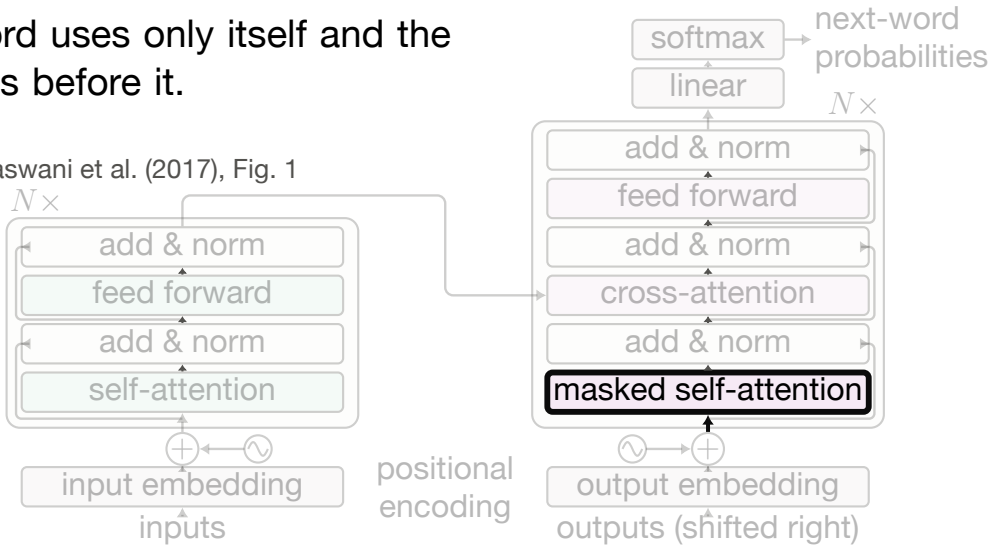
Rows: English words (queries).
Columns: French words (keys).

Each row sums to 1: the weights of an average of the French words' values.

The decoder: masked self-attention

A word uses only itself and the words before it.

after Vaswani et al. (2017), Fig. 1



The causal mask

	I	am	a	student
I	0.24	0.27	0.24	0.25
am	0.22	0.26	0.23	0.29
a	0.39	0.21	0.28	0.12
student	0.32	0.24	0.28	0.16

without the mask

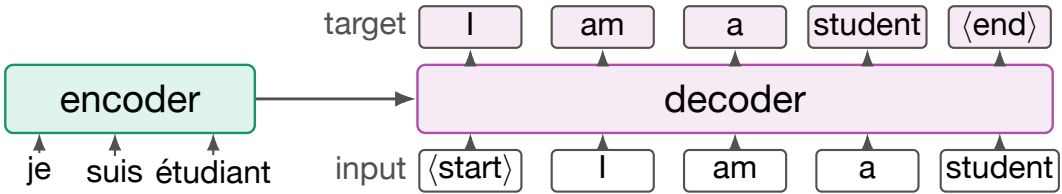
	I	am	a	student
I	1.00	0	0	0
am	0.45	0.55	0	0
a	0.44	0.24	0.32	0
student	0.32	0.24	0.28	0.16

with the causal mask

causal mask: $-\infty$ above the diagonal, before the softmax

"am" uses only "I" and "am" (0.45 and 0.55)

Training: teacher forcing

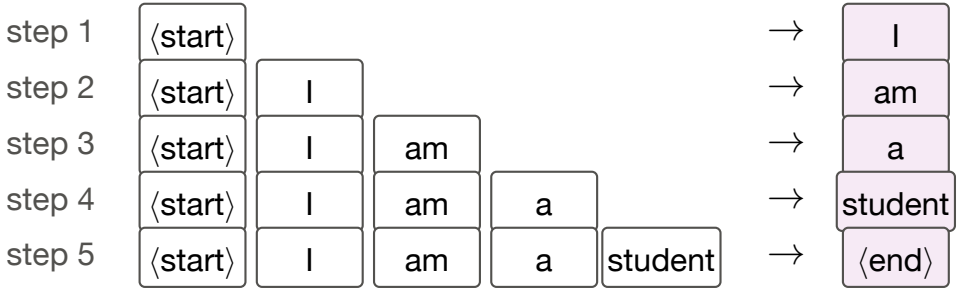


Input: ⟨start⟩ (a start token), then the correct translation.

Target: the next word at each position, then ⟨end⟩ (an end token); cross-entropy loss (Lecture 9).

Teacher forcing: the input is the correct words, not the decoder's predictions.

Prediction: one word at a time



The encoder runs once; the decoder runs once per new word.

Each step: the linear layer and the softmax give a probability for every word of the vocabulary (the set of all words the model can output); take the most probable (**greedy decoding**); stop at ⟨end⟩.

Summary: the decoder

- Masked self-attention: scores of later positions are $-\infty$ before the softmax, so position i uses positions $1, \ldots, i$.
- Cross-attention: queries from the decoder; keys and values from the encoder's output.
- Teacher forcing: the input is the correct translation with $\langle$start$\rangle$ in front (shifted right by one position); the loss is computed at every position in one pass.
- Prediction: one word per step, fed back as input, until $\langle$end$\rangle$.

Checkpoint 2

The decoder's self-attention is masked.

Why does the encoder's self-attention need no mask?

Checkpoint 2

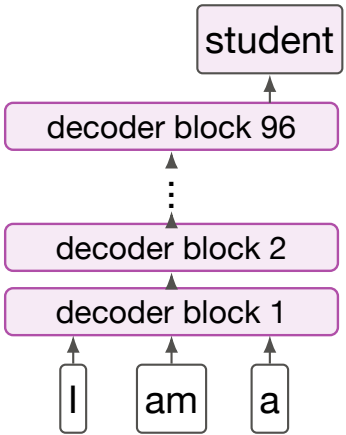
The decoder's self-attention is masked.

Why does the encoder's self-attention need no mask?

The encoder's input, the French sentence, is known in advance, so every word may use every other word.

In training, the decoder's input is the whole English sentence; without the mask, position i could use the word it has to predict.

GPT: a stack of decoder blocks



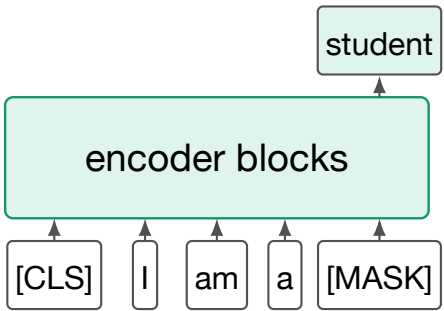
GPT (generative pre-trained transformer, OpenAI): decoder blocks without cross-attention.

Trained on a large amount of text to predict the next word, with the causal mask.

GPT-3: 96 blocks, 175 billion parameters.

Radford et al., OpenAI, 2018; Brown et al., NeurIPS 2020, Table 2.1

BERT: a stack of encoder blocks



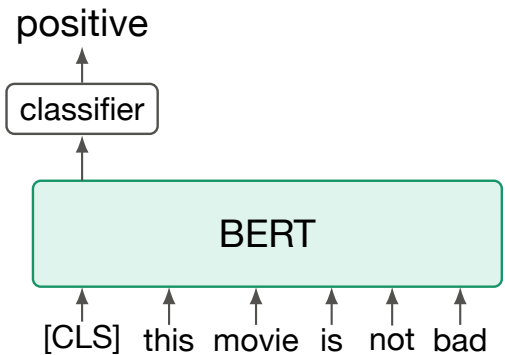
BERT (bidirectional encoder representations from transformers, Google): encoder blocks only.

No causal mask: every word uses the words on both sides.

Pre-training: choose 15% of the tokens at random, replace most by [MASK], and predict them.

Devlin et al., NAACL 2019

Fine-tuning BERT



Task: is a movie review positive or negative?

[CLS]: a special token, always first. A small classifier (linear layer and softmax) uses the output of [CLS].

Training on labeled reviews fits the classifier and updates all parameters of BERT (**fine-tuning**).

ChatGPT: three training steps

1. **Fine-tuning**: GPT is fine-tuned on answers written by people.
2. **Reward model**: a model trained on people's rankings of GPT's answers.
3. **Reinforcement learning**: GPT is updated to raise the reward model's scores.

After Ouyang et al., NeurIPS 2022, Fig. 2; OpenAI, 2022.

Summary: language models

- GPT: decoder blocks only (no cross-attention), trained to predict the next word; GPT-3 has 96 blocks.
- BERT: encoder blocks only, trained to predict hidden words from the words on both sides; fine-tuned with a classifier on [CLS].
- ChatGPT: a GPT model trained further on answers written by people, with a reward model from people's rankings, and with reinforcement learning.

Self-attention in NumPy

```
import numpy as np
def attention(Q, K, V):
    S = Q @ K.T / np.sqrt(K.shape[1])  # scaled scores
    A = np.exp(S - S.max(axis=1, keepdims=True))
    A = A / A.sum(axis=1, keepdims=True)  # softmax by row
    return A @ V, A
# Q, K, V of "I am a student" (Step 1, rounded)
Q = np.array([[-0.3, 0.1, 0.2], [0.0, 0.0, 0.4],
              [-0.3, 0.2, -1.4], [1.0, 0.5, -0.9]])
K = np.array([[0.0, 0.2, -0.2], [0.1, 0.5, 0.6],
              [0.2, -0.1, 0.1], [0.4, -0.7, 1.0]])
V = np.array([[0.4, 0.7, 0.6], [2.0, 1.1, 0.2],
              [-0.5, -1.3, -0.8], [0.0, -3.2, -2.2]])
Z, A = attention(Q, K, V)
print(A[0].round(2))  # [0.24 0.27 0.24 0.25]
print(Z.shape)        # (4, 3)
```

Row 1 of A: the weights of “I” in Step 3.

References

- J. Alammar. *The Illustrated Transformer*, 2018. https://jalammar.github.io/illustrated-transformer/
- Bala Priya C. *Build better deep learning models with batch and layer normalization*. Pinecone, 2023. https://www.pinecone.io/learn/batch-layer-normalization/
- S. Raschka. *Understanding and coding self-attention, multi-head attention, causal-attention, and cross-attention in LLMs*, 2024. https://magazine.sebastianraschka.com/p/understanding-and-coding-self-attention
- P. von Platen. *Transformer-based encoder-decoder models*. Hugging Face blog, 2020. https://huggingface.co/blog/encoder-decoder
- OpenAI. *Introducing ChatGPT*, 2022.

References: papers

- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin. *Attention is all you need*. NIPS 2017.
- A. Radford, K. Narasimhan, T. Salimans and I. Sutskever. *Improving language understanding by generative pre-training*. OpenAI, 2018.
- T. B. Brown et al. *Language models are few-shot learners*. NeurIPS 2020.
- J. Devlin, M.-W. Chang, K. Lee and K. Toutanova. *BERT: pre-training of deep bidirectional transformers for language understanding*. NAACL 2019.
- L. Ouyang et al. *Training language models to follow instructions with human feedback*. NeurIPS 2022.