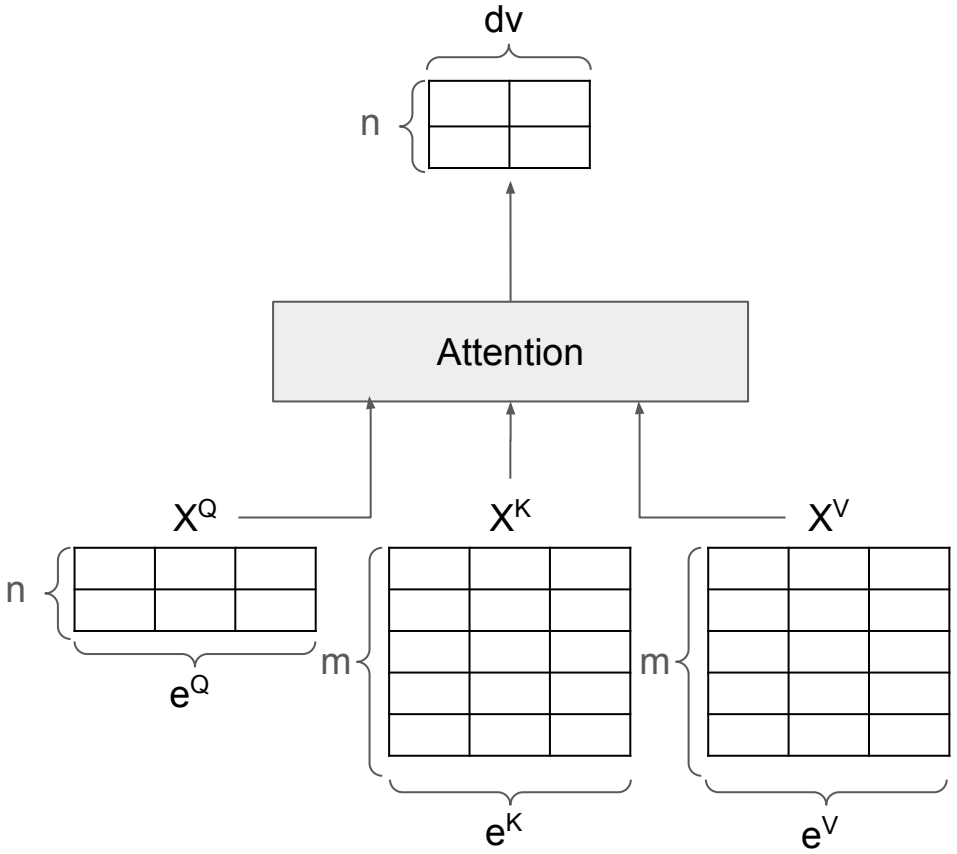
Attention

Attention przyjmuje trzy wejścia 2D $\{\mathbb{R}^{n\times eq}, \mathbb{R}^{m\times ek}, \mathbb{R}^{m\times ev}\}$ i zwraca pojedyncze wyjście 2D $\{\mathbb{R}^{n\times dv}\}$.

Jak za chwilę zobaczymy, attention przypomina sumę ważoną. Każde *query* $x^Q \in X^Q$ jest dopasowywane do wszystkich *keys* X^K. Te x^K, które lepiej pasują dokładają więcej *values* x^V do wyniku.

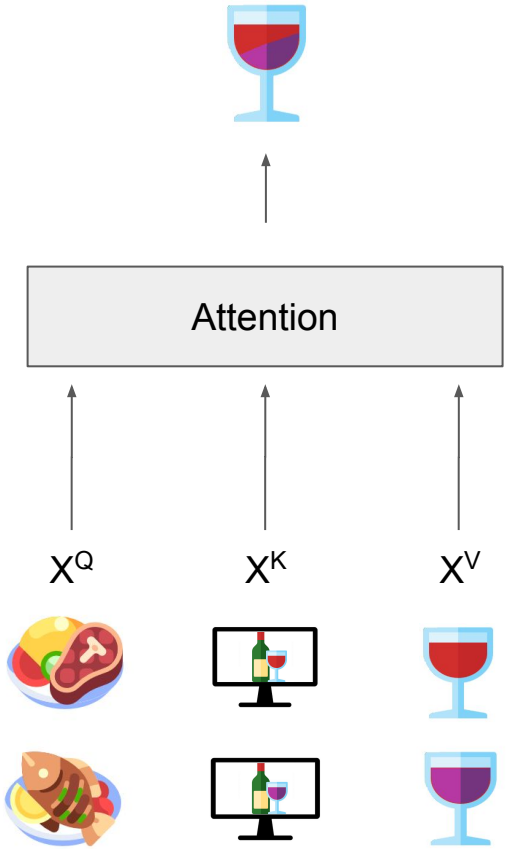
Zazwyczaj $e^Q = e^K = e^V$.



Attention: przykład

Powiedzmy, że prowadzimy restaurację. Chcemy podawać wyjątkowe wina pasujące do pozycji w naszym menu, ale dysponujemy jedynie małym zbiorem win z Lidla.

Kiedy klient prosi o wino pasujące do dania, porównujemy przepis na to danie X^Q do informacji o naszych winach znalezionych w internecie X^K. Następnie zlewamy razem wszystkie wina X^V, biorąc więcej tych, które zdawały się pasować lepiej.



https://www.freepik.com/icon

Attention: implementacja (dot-product attention)

$$\text{Attention}(X^Q, X^K, X^V) =$$

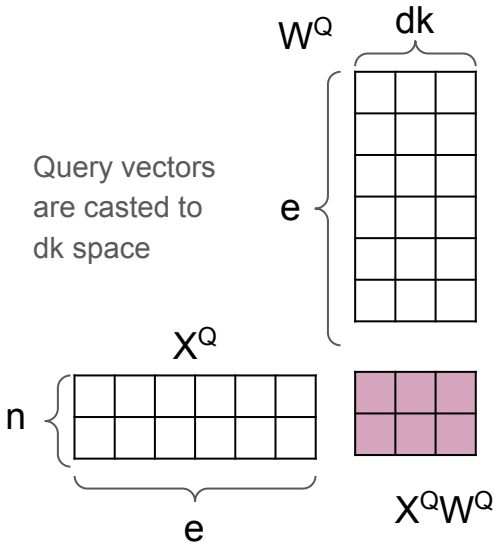
$$\text{softmax}\left(\frac{X^Q W^Q (X^K W^K)^T}{\sqrt{dk}}\right) X^V W^V$$

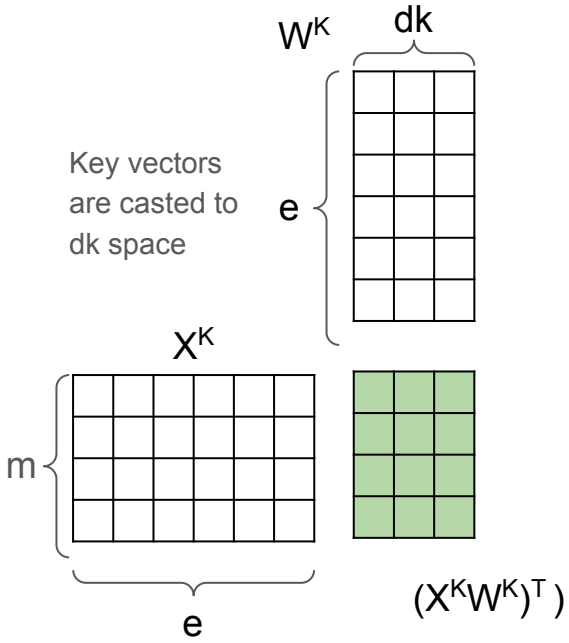
Dzielenie przez $\sqrt{dk}$ czyni wariancję argumentu softmax równą jeden, co pomaga przy trenowaniu.

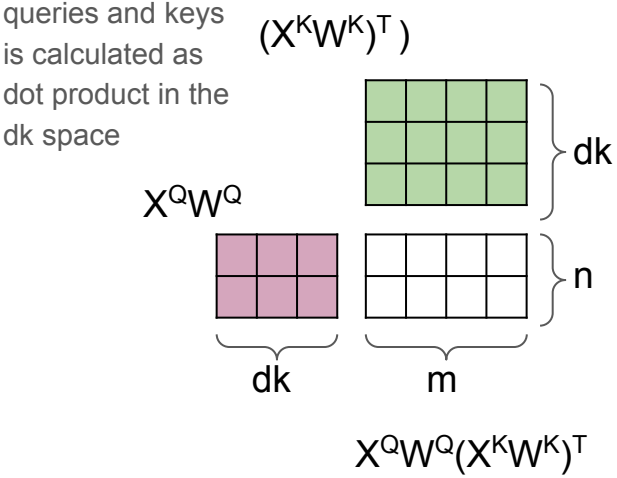
n : number of query tokens
m : number of key/value tokens
e : embedding length
dk : dimension of key representation
dv : dimension of value representation
$X^Q \in \mathbb{R}^{n \times e}$: queries
$X^K \in \mathbb{R}^{m \times e}$: keys
$X^V \in \mathbb{R}^{m \times e}$: values
$W^Q \in \mathbb{R}^{e \times dk}$: query weights (trainable)
$W^K \in \mathbb{R}^{e \times dk}$: key weights (trainable)
$W^V \in \mathbb{R}^{e \times dv}$: value weights (trainable)

Attention: query-key matching

$$\text{softmax}\left(\frac{X^Q W^Q (X^K W^K)^T}{\sqrt{dk}}\right) X^V W^V$$







Attention: czy oba W^Q i W^K są potrzebne?

$X^Q W^Q (X^K W^K)^T = X^Q W^Q (W^K)^T (X^K)^T$

$W^Q (W^K)^T$ to przekształcenie liniowe, więc zamiast trenować osobno W^Q i W^K, moglibyśmy użyć jednej macierzy W^{QK}.

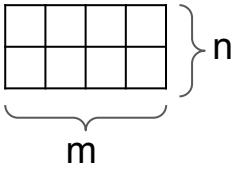
Jednakże, $W^Q \in \mathbb{R}^{e \times dk}$, $W^K \in \mathbb{R}^{e \times dk}$, więc $W^{QK} \in \mathbb{R}^{e \times e}$

W oryginalnym *Transformer* z 2017, e=512, dk=64, więc użycie W^{QK} zwiększyłoby liczbę parametrów czterokrotnie. Prawdziwy powód, dla którego używamy obu W^Q i W^K to redukcja złożoności.

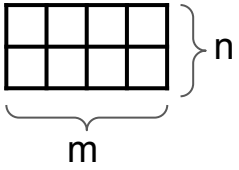
Attention: suma ważona

$$\text{softmax}\left(\frac{X^Q W^Q (X^K W^K)^T}{\sqrt{dk}}\right) X^V W^V$$

$$\frac{X^Q W^Q (X^K W^K)^T}{\sqrt{dk}}$$

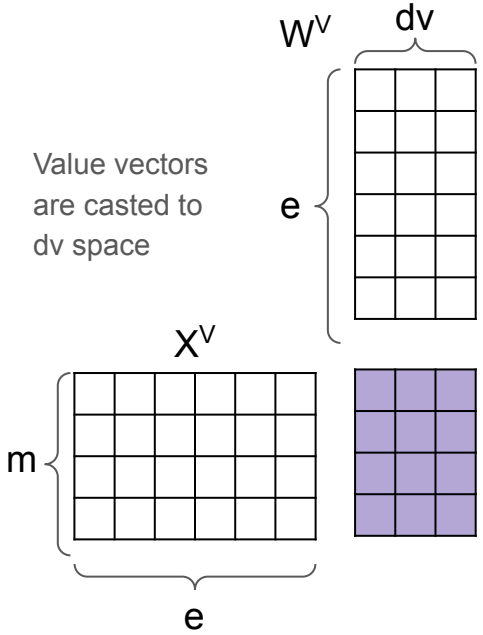


Every row sums up to one after softmax



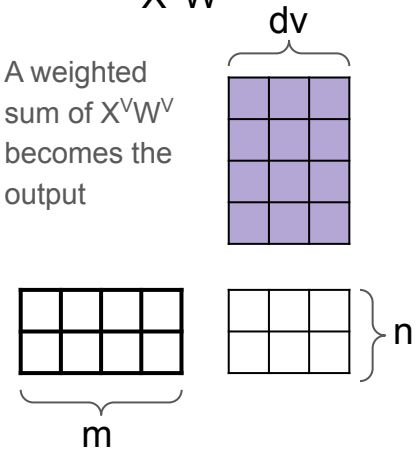
$$\text{softmax}\left(\frac{X^Q W^Q (X^K W^K)^T}{\sqrt{dk}}\right)$$

Value vectors are casted to dv space



$X^V W^V$

A weighted sum of $X^V W^V$ becomes the output



Attention: kolejność tokenów na wejściu nie ma znaczenia

$$\text{Attention}(X^Q, X^K, X^V) = \text{softmax}\left(\frac{X^Q W^Q (X^K W^K)^T}{\sqrt{dk}}\right) X^V W^V$$

Permutacje X^Q zmieniają jedynie kolejność wyjść, a permutacje (X^K, X^V) nie zmieniają nic.

Sprzeczne z popularnym przekonaniem: skoro attention jest stosowane na ciągach tokenów, to na pewno zna ich kolejność?

Okazuje się, że nie: jeśli pozycja tokenu jest ważna, musi być w nim zapisana.

X^Q to zbiór, a (X^K, X^V) to zbiór par.

Multi-head attention

Wiele modułów attention przyjmujących te same dane; ich wyniki są konkatenowane.

$$\text{MultiheadAttention}(X^Q, X^K, X^V) =$$

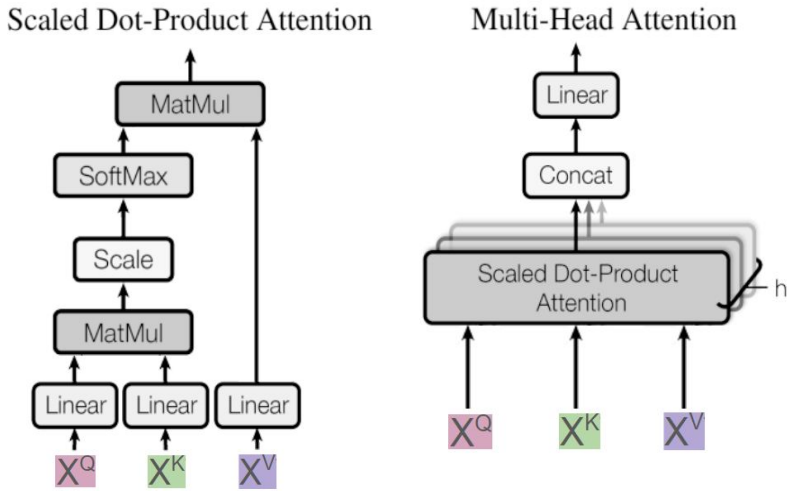
$$\underset{i \in \{1, h\}}{\text{concat}} \left(\text{softmax}\left(\frac{X^Q W_i^Q (X^K W_i^K)^T}{\sqrt{dk}} \right) X^V W_i^V \right) W^O$$

W typowych implementacjach Transformera, dk=dv=e/h, do=e.

h: number of heads
n : number of query tokens
m : number of key/value tokens
e : embedding length
dk : dimension of key representation
dv : dimension of value repr.
do : dimension of output repr.
$X^Q \in \mathbb{R}^{n \times e}$: query tokens
$X^K \in \mathbb{R}^{m \times e}$: key tokens
$X^V \in \mathbb{R}^{m \times e}$: value tokens
$W_i^Q \in \mathbb{R}^{e \times dk}$: query weights
$W_i^K \in \mathbb{R}^{e \times dk}$: key weights
$W_i^V \in \mathbb{R}^{e \times dv}$: value weights
$W^O \in \mathbb{R}^{dv*h \times do}$: output weights

Multi-head attention: diagram

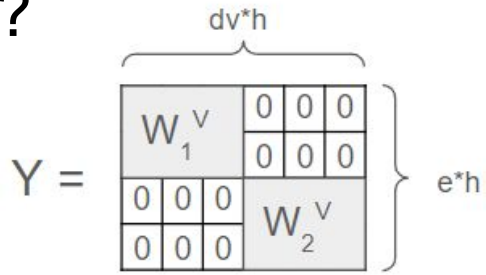
$$\underset{i \in \{1,h\}}{\text{concat}} \left(\text{softmax}\left(\frac{X^Q W_i^Q (X^K W_i^K)^T}{\sqrt{dk}} \right) X^V W_i^V \right) W^O$$



Attention Is All You Need (2017)

Multi-head attention: czy W_i^V są niezbędne?

Podobnie jak W^Q and W^K, zmniejszają złożoność



$$\underbrace{\underset{i \in \{1,h\}}{\text{concat}}\left(\text{softmax}\left(\frac{X^Q W_i^Q (X^K W_i^K)^T}{\sqrt{dk}}\right) X^V W_i^V\right) W^O}_{\in \mathbb{R}^{n \times dv*h}} =$$

$$W_i^V \in \mathbb{R}^{e \times dv} \quad W^O \in \mathbb{R}^{dv*h \times do}$$
$$Y \in \mathbb{R}^{e*h \times dv*h} \quad YW^O \in \mathbb{R}^{e*h \times do}$$

$$\underbrace{\underset{i \in \{1,h\}}{\text{concat}}\left(\text{softmax}\left(\frac{X^Q W_i^Q (X^K W_i^K)^T}{\sqrt{dk}}\right) X^V\right) YW^O}_{\in \mathbb{R}^{n \times e*h}}$$

YW^O można byłoby zastąpić macierzą $Z \in \mathbb{R}^{e*h \times do}$, ale to zmieniłoby liczbę parameterów z h*e*dv+h*dv*do na e*h*do. W oryginalnym Transformer to h/2-krotny wzrost.

Warianty attention

W praktyce najczęściej używane są trzy ograniczone warianty *attention*.

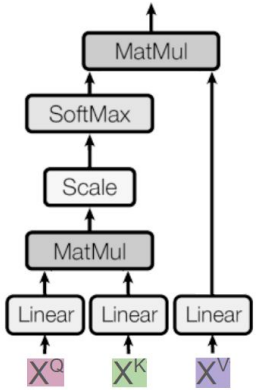
Cross-attention: $X^Q \neq X^K = X^V$

Self-attention: $X^Q = X^K = X^V$

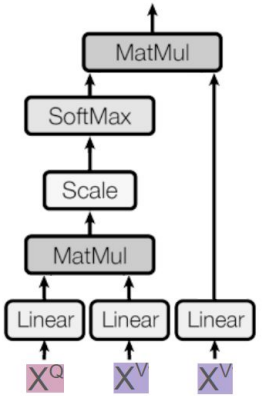
Causal self-attention: $X^Q = X^K = X^V$ & "wcześniejsze" tokeny nie dopasowują się do "późniejszych".

Trzy architektury: *encoder-only*, *decoder-only* i *encoder-decoder*.

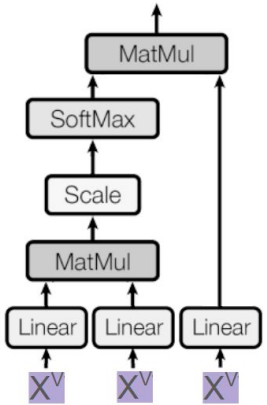
attention



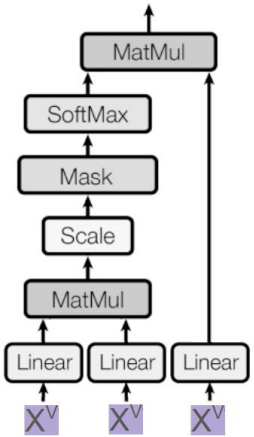
cross-attention



self-attention



causal self-attention



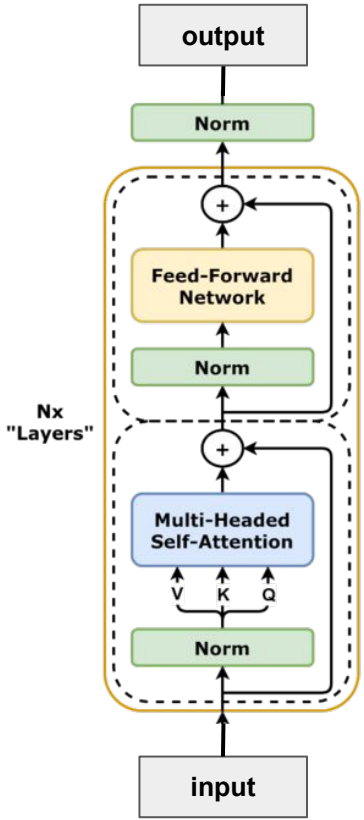
Attention Is All You Need (2017)

Encoder-only Transformer

Sieć używająca self-attention,
fully-connected layers, residual connections
i layer normalization.

FC operują na pojedynczych tokenach.

BERT(liczba warstw L/embedding size e)
BERT-Large(24/1024): 340M parametrów
BERT-Base(12/768): 110M parametrów
Sieci FC mają dwie warstwy ukryte: jedną wielkości 4*e i jedną wielkości e.



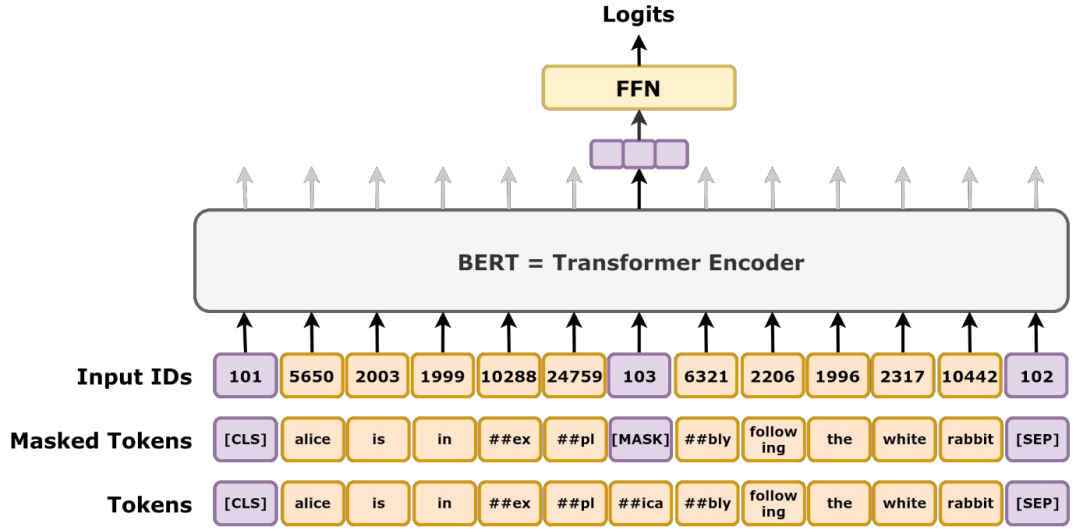
https://github.com/dvgodoy/dl-visuals/?tab=readme-ov-file

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2018)

Pre-training encoder-only Transformers

BERT jest pre-trained na dużym korpusie (3,3B słów). Jego zadanie to zamiana wszystkich tokenów [MASK] na przewidywane tokeny. Pozostałe wyjścia nie wpływają na funkcję straty.

Ucząc się wykonywać to proste zadanie, BERT poznaje zależności występujące w tekście.



use Adam with learning rate of 1e-4, $\beta_1 = 0.9$, $\beta_2 = 0.999$, L2 weight decay of 0.01, learning rate warmup over the first 10,000 steps, and linear decay of the learning rate. We use a dropout probability of 0.1 on all layers. We use a `gelu` acti-

We train with batch size of 256 sequences (256 sequences * 512 tokens = 128,000 tokens/batch) for 1,000,000 steps, which is approximately 40 epochs over the 3.3 billion word corpus.

https://dvgodoy.github.io/dl-visuals/BERT/

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2018)

Fine-tuning encoder-only Transformers

Pre-trained BERT może być fine-tuned do innego zadania, zwykle na znacznie mniejszym zbiorze treningowym. W zależności od typu zadania, niewielkie warstwy FC mogą zostać dodane na koniec.

parameter	pre-train	fine-tune
lr	1e-4	3e-5
batch size	256	16, 32
epochs	40	2, 3, 4

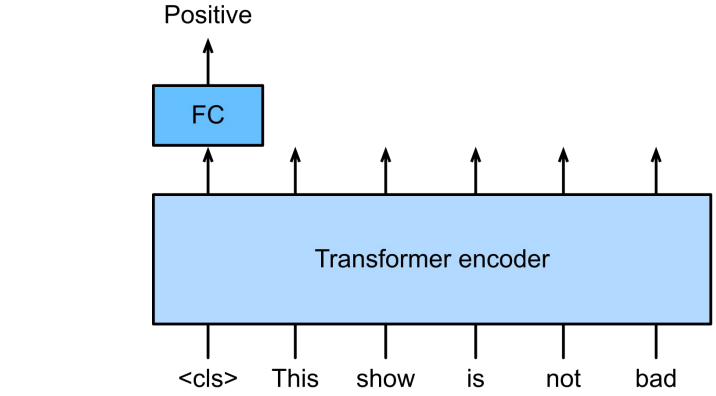
SST-2 The Stanford Sentiment Treebank is a binary single-sentence classification task consisting of sentences extracted from movie reviews with human annotations of their sentiment (Socher et al., 2013).

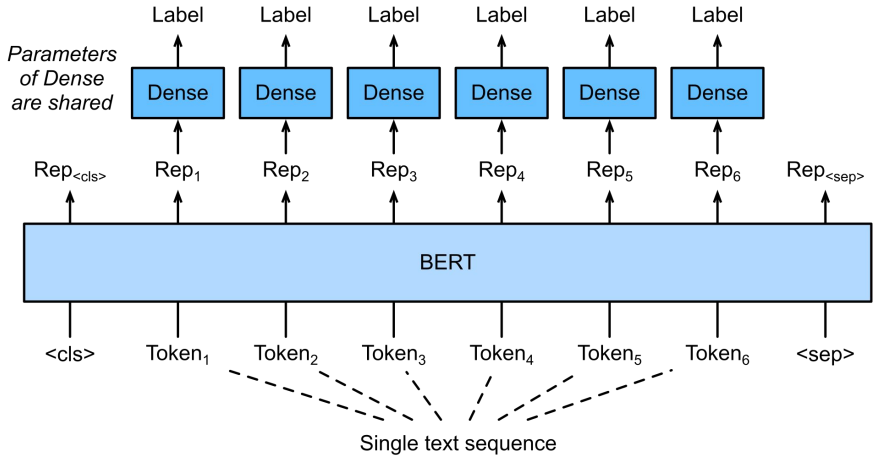
CoLA The Corpus of Linguistic Acceptability is a binary single-sentence classification task, where the goal is to predict whether an English sentence is linguistically "acceptable" or not (Warstadt et al., 2018).

Fine-tuning: przykłady

Klasyfikacja: zamiast opierać się na całym wyjściu, klasyfikacja bazuje jedynie na wyjściu tokena specjalnego <cls>.

Part-of-speech tagging: wszystkie wyjścia są używane do przewidzenia części mowy odpowiadających poszczególnym tokenom.

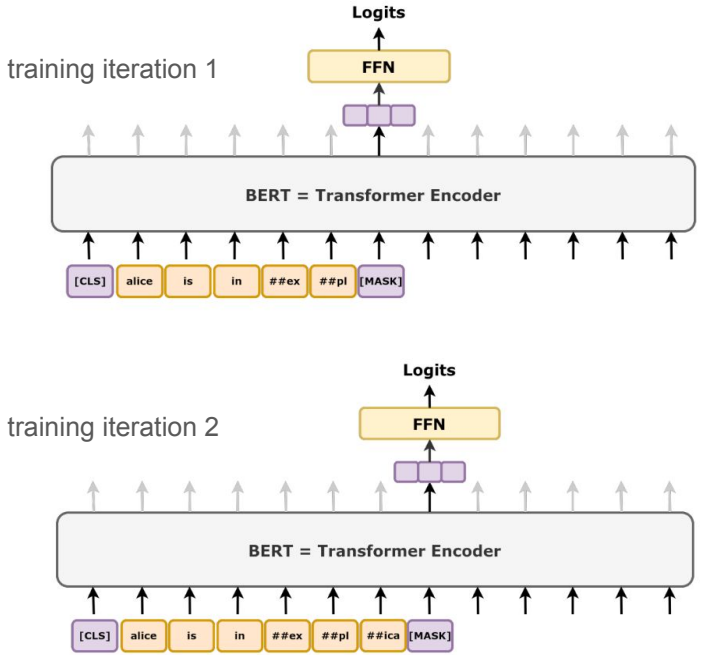




Encoder-only Transformers kiepsko uczą się generacji

W teorii, encoder Transformer może być użyty do generacji tekstu; wystarczy dodać token [MASK] na końcu. W praktyce, przy standardowym *pre-training* prowadzi to do kiepskich wyników z powodu distribution shift.

Z drugiej strony, *pre-training* jedynie na tekstach kończących się na [MASK] jest zbyt kosztowny, bo wtedy tylko jeden token może być ewaluowany podczas jednej iteracji.



https://github.com/dvgodoy/dl-visuals/?tab=readme-ov-file

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2018)

Causal self-attention

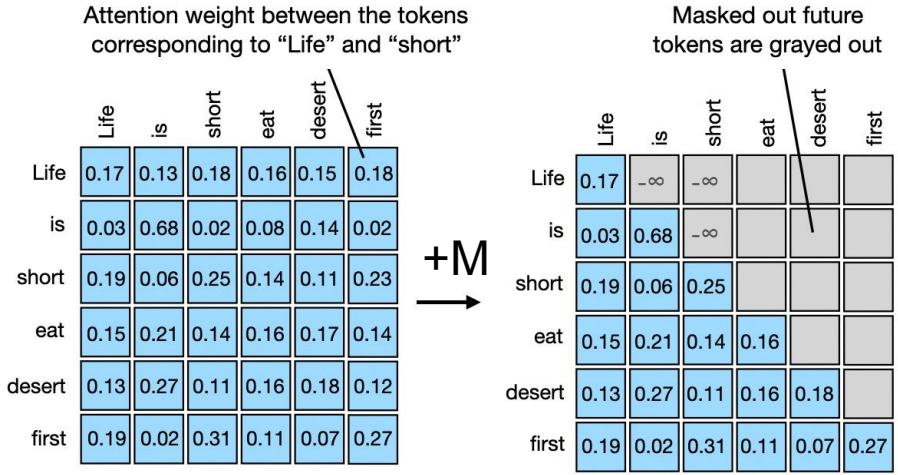
Modyfikacja attention, która ułatwia uczenie generowania.

Zamiast maskować wejścia, argumenty softmax są maskowane: każdy token może dopasować się tylko do siebie i poprzednich tokenów.

W trakcie trenowania wiele predykcji może być ewaluowanych jednocześnie.

Wprowadzona jest kolejność tokenów i jednokierunkowy przepływ informacji.

$$\text{CausalAttention}(X^Q, X^K, X^V) = \text{softmax}\left(\frac{X^Q W^Q (X^K W^K)^T}{\sqrt{dk}} + M\right) X^V W^V$$



https://magazine.sebastianraschka.com/p/understanding-and-coding-self-attention

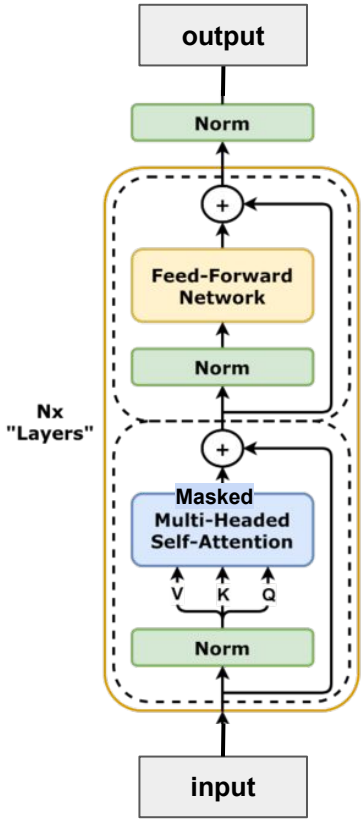
Decoder-only Transformer

To samo, co encoder-only Transformer, tylko z *causal (a.k.a. masked) self-attention*.

Pozwala to na znacznie szybsze uczenie, ale może pogarszać wyniki.

Llama3: 8B, 70B, 405B parametrów

	8B	70B	405B
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	14,336	28,672	53,248
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	3×10^{-4}	1.5×10^{-4}	8×10^{-5}
Activation Function	SwiGLU		
Vocabulary Size	128,000		
Positional Embeddings	RoPE ($\theta = 500,000$)		



warstwy FC operują na pojedynczych tokenach, więc nie wymagają maskowania

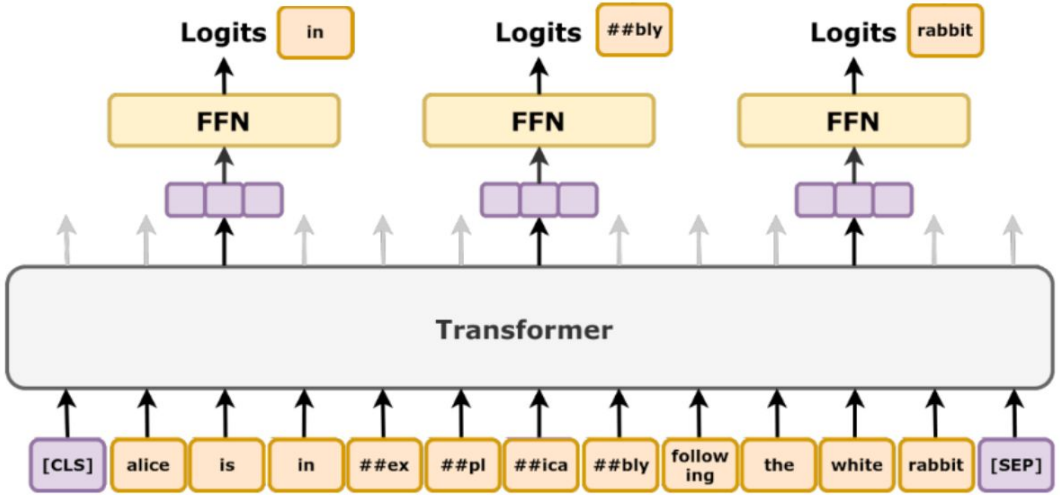
https://github.com/dvgodoy/dl-visuals/?tab=readme-ov-file

The Llama 3 Herd of Models (2024)

Pre-training decoder-only Transformers

Przewidywanie następnego tokenu. Masked attention gwarantuje, że przewidywanie tokenu korzysta tylko z tokenów przed nim. Ręczne maskowanie nie jest wymagane.

Llama3: 15.6T tokens. Uczenie 405B zajęło 54 dni na 16K H100 GPU. AdamW, lr 3e-4, lr schedule. Tylko jedna epoka (wyjątek: Wikipedia i BookCrawl, gdzie dwie).



https://dvgodoy.github.io/dl-visuals/BERT/

The Llama 3 Herd of Models (2024)

Encoder-decoder Transformer

Jedno stałe wejście jest analizowane przez encoder. Drugie wejście jest analizowane i generowane przez decoder. Architektura dobra do seq2seq: tłumaczenia, odpowiadanie na pytania, streszczenia. Wolniejsza niż decoder-only i przez to mniej popularna. Jej przewaga nad decoder-only zdaje się maleć ze skalą.

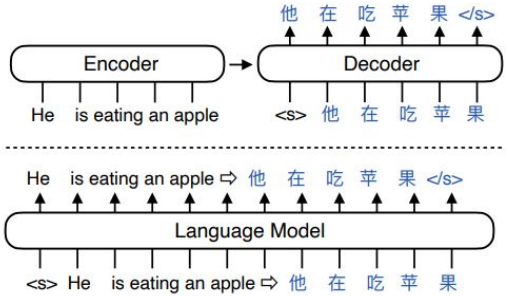
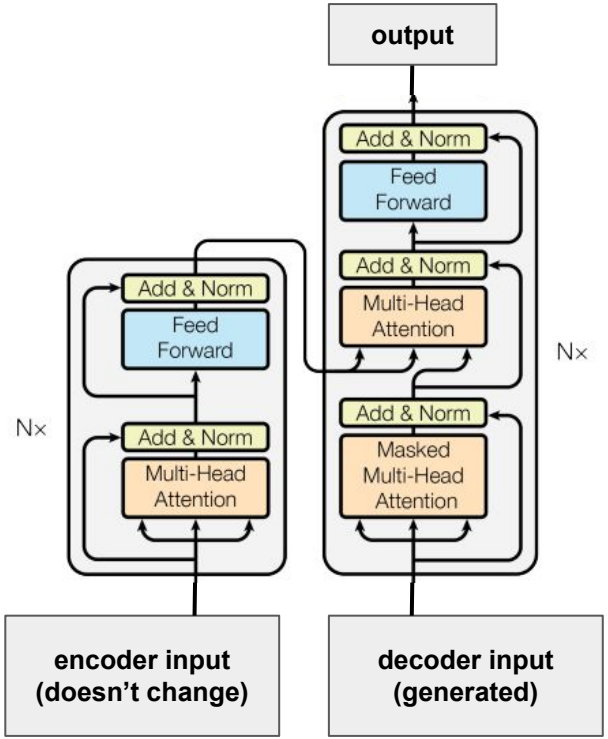


Figure 1: Encoder-Decoder (ED) framework and decoder-only Language Model (LM).



Attention Is All You Need (2017)

Decoder-Only or Encoder-Decoder? Interpreting Language Model as a Regularized Encoder-Decoder (2023)

Pre-training encoder-decoder Transformers

T5 było uczone na przewidywaniu usuniętych fragmentów tekstu.

Dane: CommonCrawl, 34B tokenów.

T5: ~2.8B parametrów, L=24, e=1024, FC layer: 16384, h=32.

Może być fine-tuned do każdego zadania sentence-to-sentence, np. tłumaczenie, generacja, pisanie podsumowań itd.

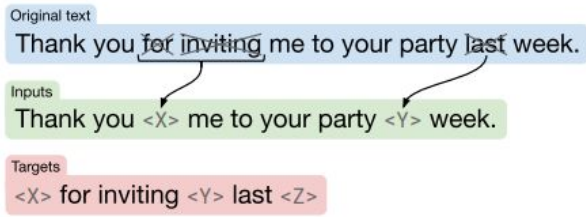


Figure 2: Schematic of the objective we use in our baseline model. In this example, we process the sentence "Thank you for inviting me to your party last week." The words "for", "inviting" and "last" (marked with an ×) are randomly chosen for corruption. Each consecutive span of corrupted tokens is replaced by a sentinel token (shown as <X> and <Y>) that is unique over the example. Since "for" and "inviting" occur consecutively, they are replaced by a single sentinel <X>. The output sequence then consists of the dropped-out spans, delimited by the sentinel tokens used to replace them in the input plus a final sentinel token <Z>.

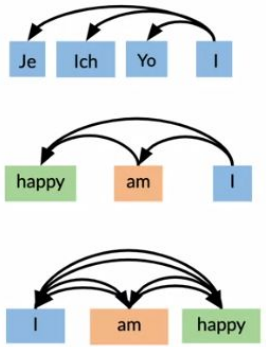
Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer (2019)

Transformer: podsumowanie architektur

Architektura	encoder-only	decoder-only	encoder-decoder
Attention	self-	masked self-	self-, masked self-, cross-
Zastosowanie*	analiza	generacja	tłumaczenie
Przykłady (parametry)	BERT (1.3B)	GPT-4 (1.8T?), Llama (405B)	T5 (2.8B), BART (406M)

Three ways of attention

- **Encoder/decoder attention**: One sentence (decoder) looks at another one (encoder)
- **Causal (self) attention**: In one sentence, words look at previous words (used for generation)
- **Bi-directional self attention**: In one sentence, words look at both previous and future words



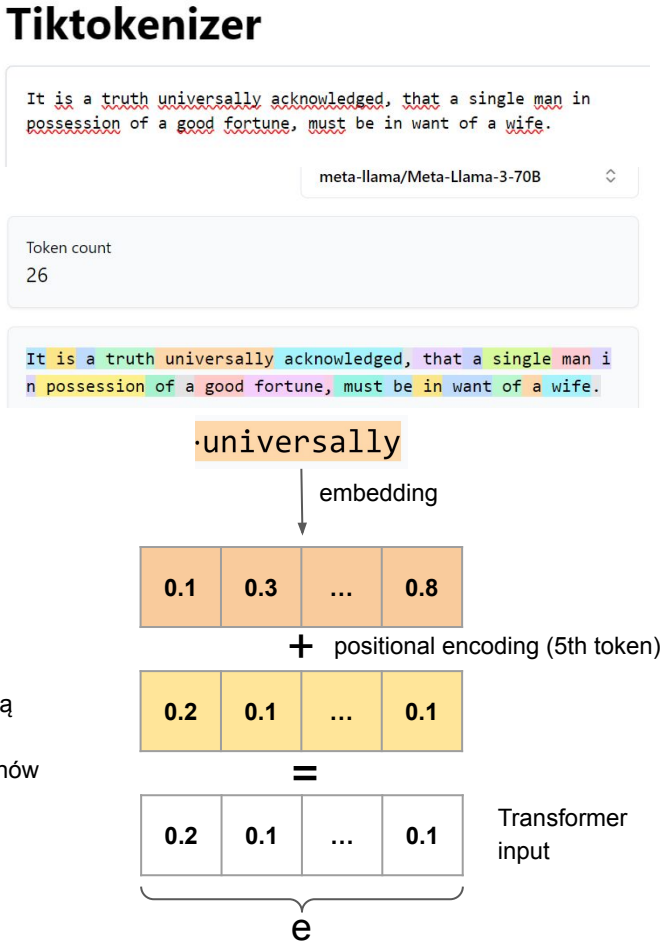
https://blog.sailor.plus/deep-learning/attention/

* nie wyłącznie: można używać encoder-decoders do generacji, decoders do analizy itd.

Przekazywanie tekstu

Transformery przyjmują nieuporządkowane (z wyjątkiem causal self-attention) zbiory wektorów. Każde dane muszą być zamienione na ten format.

Text: tokenization + embedding + positional encoding. Embedding to przekształcenie liniowe $\mathbb{R}^{t \times e}$, gdzie t to rozmiar alfabetu.



universally

embeddings, choć generowane za pomocą pojedynczego przekształcenie liniowego, zawierają informacje na temat relacji tokenów

Word	universally	uniformly	explicitly	widely	invariably	duly
Cosine Similarity	1.00	0.73	0.70	0.69	0.69	0.65

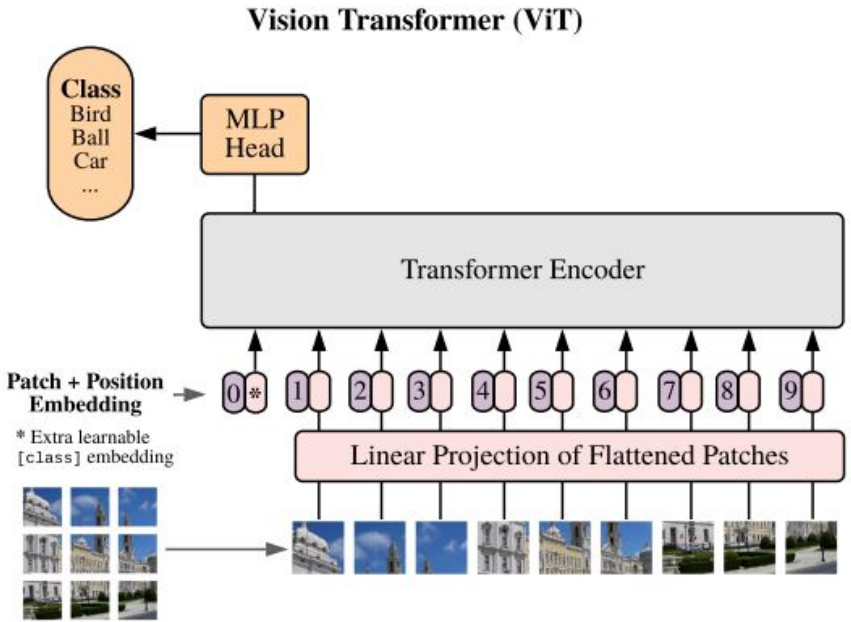
https://tiktokenizer.vercel.app/?model=meta-llama%2FMeta-Llama-3-70B

Przekazywanie obrazu

Podobne do przekazywania tekstu: obraz dzielimy na kawałki + embedding + positional encoding.

Positional encoding może zawierać info o położeniu w 2D, ale 1D działa okej.

Vision Transformer to encoder-only Transformer trenowany do klasyfikacji (86M parametrów, potem powiększony do 22B).



An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (2021)
Scaling vision transformers to 22 billion parameters (2023)