# Wprowadzenie do transformatorów... czyli kilka słów o mechanizmie atencji

Kamil Książek

Wydział Matematyki i Informatyki
Uniwersytet Jagielloński
Prof. Stanisława Łojasiewicza 6
30-348 Kraków, Poland
kamil.ksiazek@uj.edu.pl

group of machine learning research

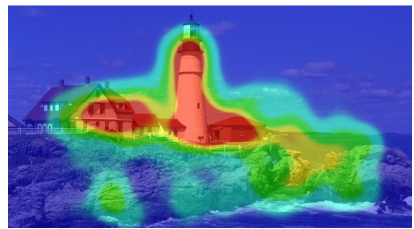
23 stycznia 2025

# Agenda

- Wprowadzenie
- Definicja atencji
- Atencja wielogłowicowa
- Warstwa normalizacyjna
- Wektory pozycyjne
- Transformator wizyjny

# Mechanizm atencji

- Mechanizm atencji (ang. *attention*) zaczął być stosowany w wizji komputerowej w 2014 roku, aby wskazać ważne regiony obrazu, które przyczyniają się do otrzymania pożądanego wyjścia. Ułatwia to użytkownikowi zrozumienie, gdzie i na czym skupiony jest model.
- W 2017 roku pojawiła się praca *Attention Is All You Need* autorstwa Ashisha Vaswaniego i in., która wprowadzała architekturę transformatorów (ang. transformers) z użyciem mechanizmu atencji.



Źródło: P. Poupart, CS480/680 Lecture 19: Attention and Transformer Networks, Spring 2019, University of Waterloo.

# Sieci rekurencyjne vs transformatory

Wyzwania związane z sieciami rekurencyjnymi:

- długodystansowe zależności;
- eksplozja i zanik gradientów;
- duża liczba kroków treningowych;
- rekurencja uniemożliwia zrównoleglenie obliczeń.

Zalety transformatorów:

- lepiej radzą sobie z długodystansowymi zależnościami;
- brak eksplozji i zaniku gradientów;
- mniejsza liczba kroków treningowych;
- brak rekurencji umozliwia zrównoleglenie obliczeń.

# Definicja atencji

Mechanizm atencji naśladuje odzyskanie wartości $\boldsymbol{v_i}$ (ang. *value*) dla zapytania $\boldsymbol{q}$ (ang. *query*) w oparciu o klucz $\boldsymbol{k_i}$ (ang. *key*) z bazy danych.

Niech $T$ będzie liczbą kluczy i odpowiadających im wartości w bazie, a $f(q, k_i)$ funkcją podobieństwa, gdzie $i \in \{1, ..., T\}$. Wowczas

$$Attention(q, \boldsymbol{k}, v) = f(q, k_1) \cdot v_1 + f(q, k_2) \cdot v_2 + ... + f(q, k_T) \cdot v_T \quad (1)$$

# Funkcje podobieństwa

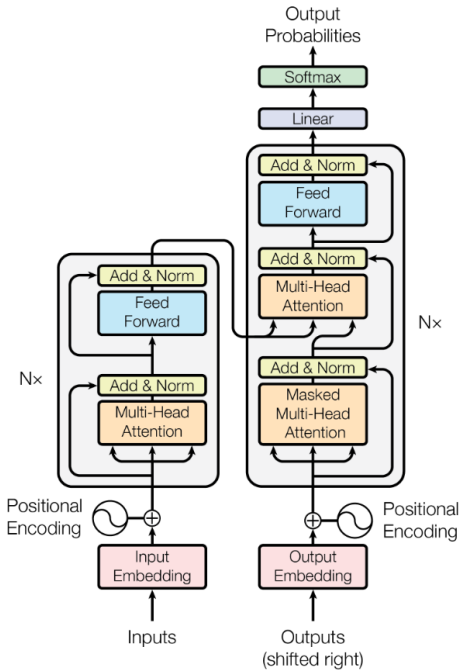
Mierzymy podobieństwo między zapytaniem a poszczególnymi kluczami od 1 do $T$.

$$s_i = f(q, k_i) = \begin{cases} q^\top k_i & \text{iloczyn skalarny (ang. } \textit{dot product}\text{)} \\ \frac{q^\top k_i}{\sqrt{d}} & \text{przeskalowany iloczyn skalarny (ang. } \textit{scaled dot product}\text{)} \\ q^\top W k_i & \text{uogólniony iloczyn skalarny (ang. } \textit{general dot product}\text{)} \end{cases}$$

gdzie $d$ jest wymiarowością klucza, zaś $W$ jest macierzą wag optymalizowaną przez sieć.

Na policzone wartości funkcji podobieństwa nakładana jest funkcja softmax, tj.:

$$a_i = \frac{e^{s_i}}{e^{s_1} + e^{s_2} + ... + e^{s_T}} \tag{2}$$




## Architektura transformatora

(A. Vaswani et al, Attention Is All You Need, Proceedings of NeurIPS 2017.)

# Multi-head attention

Atencja wielogłowicowa (ang. multi-head attention) sprowadza się do utworzenia wielu głowic atencji (dla zapytania) z różnymi wagami. W ten sposób łączymy informacje z różnych reprezentacji.

Niech $h$ będzie liczbą głowic atencji, $W^O$, $W_i^Q$, $W_i^K$, $W_i^V$ macierzami wag. Wówczas:

$$MultiHead(Q, K, V) = Concat(head_1, head_2, ..., head_h)W^O. \tag{3}$$

Pojedyncza głowica obliczana jest następująco:

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V), \tag{4}$$

gdzie $i \in \{1, ..., h\}$, z kolei

$$Attention(Q, K, V) = Softmax\left(\frac{Q^\top K}{\sqrt{d}}\right)V. \tag{5}$$

Autorzy pracy *Attention Is All You Need* używają 6 bloków w enkoderze i dekoderze, $d = 64$, $h = 8$.

# Masked multi-head attention

- Zamaskowana atencja wielogłowicowa (ang. masked multi-head attention) dotyczy sytuacji, w której część wartości zostaje zamaskowana, tzn. prawdopodobieństwo wyboru takich wartości wynosi 0.
- Podczas dekodowania wartość wyjściowa zależy tylko od wcześniejszych, a nie przyszłych wyjść.

$$Attention(Q, K, V) = Softmax\left(\frac{Q^\top K}{\sqrt{d}}\right)V \tag{6}$$

W zamaskowanej atencji, w liczniku argumentu funkcji Softmax dodajemy macierz $M$ składającą się z wartości 0 oraz $-\infty$, tj.:

$$MaskedAttention(Q, K, V) = Softmax\left(\frac{Q^\top K + M}{\sqrt{d}}\right)V. \tag{7}$$

# Normalizacja warstwy

- Wykonywana jest dla całej warstwy, a nie dla próbki danych, jak w przypadku normalizacji wsadowej (ang. batch normalization).
- Normalizuje wartości wejściowe do neuronów w danej warstwie, by miały średnią 0 oraz wariancję równą 1.
- Dla $i$-tej jednostki ukrytej, gdzie $i \in \{1, ..., H\}$, wyliczane jest:

$$LN(h_i) = \frac{g_i}{\sigma}(h_i - \mu), \tag{8}$$

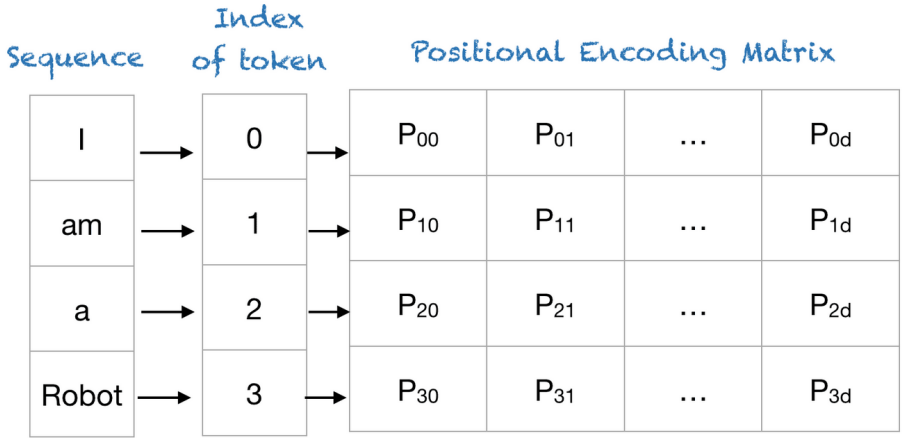
gdzie $g_i$ jest parametrem zysku (ang. gain parameter), $\mu = \frac{1}{H}\sum_{i=1}^{H} h_i$, zaś

$\sigma = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(h_i - \mu)^2}$.

J. Xu i in., Understanding and Improving Layer Normalization, NeurIPS 2019.

# Wektory pozycyjne

- Osadzone wektory pozycyjne (ang. positional embeddings) mają za zadanie opisać pozycję danego elementu w ciągu w sposób jednoznaczny.




Źródło: machine-learningmystery.com

Positional Encoding Matrix for the sequence 'I am a robot'

# Wektory pozycyjne

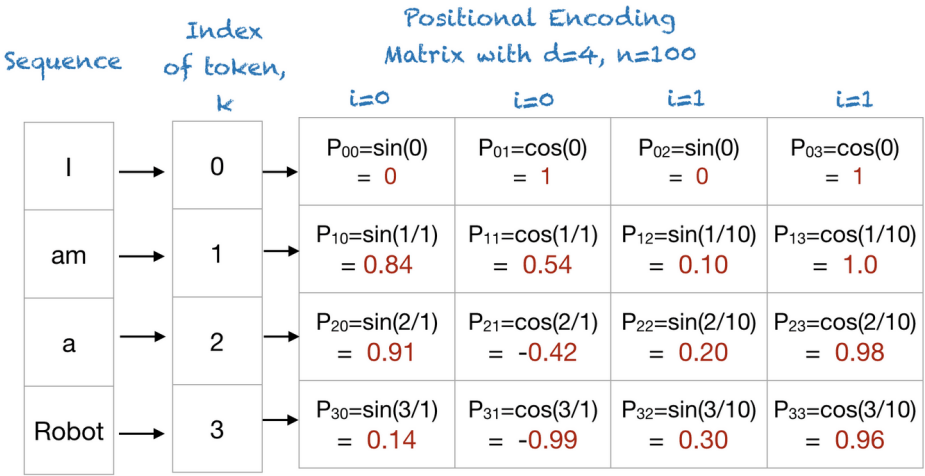
- Przypuśćmy, że nasz ciąg wejściowy ma wymiar $L$ i chcemy określić reprezentację $k$-tej pozycji tego ciągu, gdzie $k \in \{0, 1, ..., L-1\}$. Wówczas

$$P(k, 2i) = \sin\left(\frac{k}{n^{\frac{2i}{d}}}\right), \tag{9}$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{\frac{2i}{d}}}\right); \tag{10}$$

gdzie $d$ jest wymiarem wektora pozycyjnego, $P(k, j)$ odwzorowuje $k$-tą pozycję wejściowego ciągu do elementu znajdującego się w pozycji $(k, j)$ macierzy pozycyjnej, $n$ jest parameterem (w pracy *Attention is All You Need* określonym na 10000), $i \in \{0, 1, ..., \frac{d}{2}\}$ określa kolumnę macierzy pozycyjnej. Ustalone $i$ odwzorowuje element z użyciem sinusa oraz cosinusa.

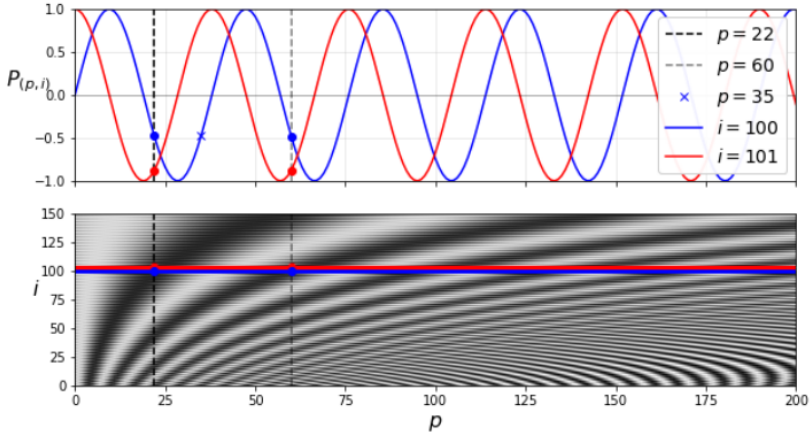
# Wektory pozycyjne




Positional Encoding Matrix for the sequence 'I am a robot'

Źródło: machinelearningmystery.com

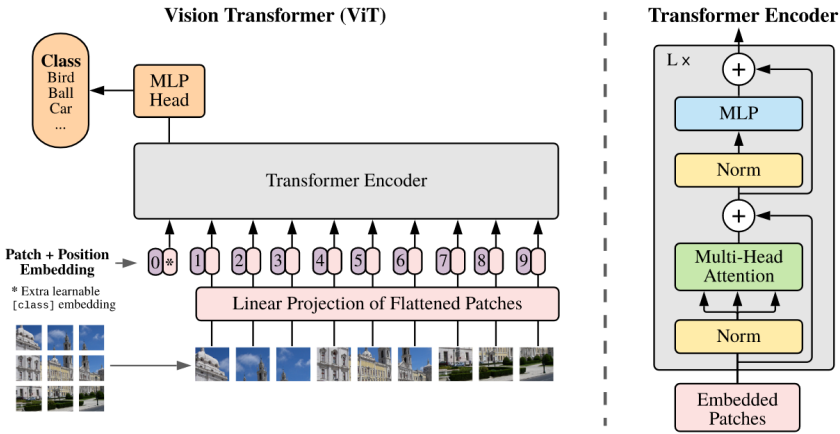
# Wektory pozycyjne




Macierz sinusoidalnych oraz cosinusoidalnych kodowań pozycyjnych. $p$ oznacza pozycję słowa w zdaniu, $i$ oznacza kolejny wymiar kodowania, z kolei $P(p, i)$ wartość kodowania $p$-tego słowa w $i$-tym wymiarze.

Źródło: A. Geron, Uczenie maszynowe z użyciem Scikit-Learn, Keras i TensorFlow, Wydanie III, Helion 2023; https://github.com/ageron/handson-ml3/

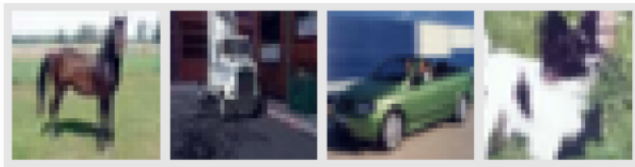
# Transformator wizyjny (ang. Vision Transformer, ViT)




Źródło: A. Dosovitskiy i in., An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale, ICLR 2021.

# Podział obrazu na fragmenty (ang. patches)

- Przykładowe obrazy zbioru CIFAR-10:



- Powyższe obrazy jako ciągi fragmentów:

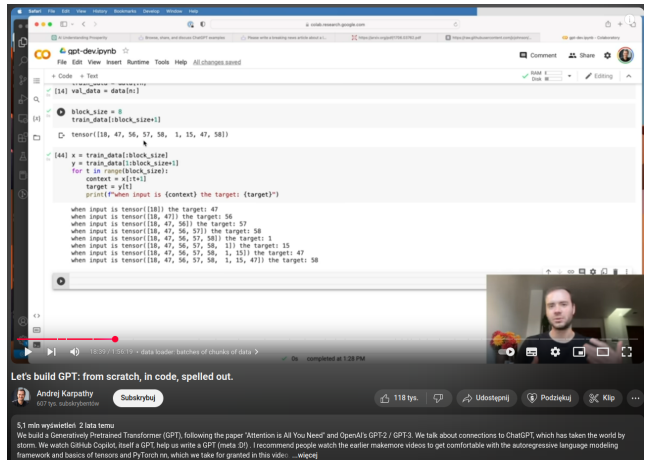
Źródło: Phillip Lippe, uvadlc-notebooks.readthedocs.io.

# Moduły transformatora wizyjnego

- Warstwa **rzutu liniowego** przekształca wejściowe fragmenty obrazu do wektora cech o większym rozmiarze. Jest to liniowa warstwa, która niezależnie przyjmuje na wejściu każdy z fragmentów.
- **Token klasyfikujący** (CLS), który jest dołączany do przetworzonych fragmentów obrazu i ma za zadanie *reprezentować cały obraz*, co dalej może posłużyć do klasyfikacji.
- Trenowalne **wektory pozycyjne**, które są dołączane do tokenów (danych wejściowych wraz z CLSem).
- Część kodująca transformatora (ang. *transformer encoder*).
- Głowa MLP.

# Rekomendacje

- Tutorial z transformatorów wizyjnych – omówienie wraz z przykładem kodu: **UvA DL Notebooks, Tutorial 15: Vision Transformers**.
- Implementacja modelu językowego minGPT z wprowadzeniem Andreja Karpathy'ego (live coding) – **Let's build GPT: from scratch, in code, spelled out**.




Źródło: youtube.com

# Dziękuję za **atencję**!

ŹRÓDŁA:


1. A. Vaswani et al, Attention Is All You Need, Proceedings of NeurIPS 2017.
2. P. Poupart, CS480/680 Lecture 19: Attention and Transformer Networks, University of Waterloo, 2019.
3. M. Saeed, A Gentle Introduction to Positional Encoding in Transformer Models, Part 1, machinelearningmystery.com, 2023.
4. P. Lippe, Tutorial 15: Vision Transformers, uvadlc-notebooks.readthedocs.io.