



Natural Language Processing

język człowieka w świecie maszyn

Agenda

1. Czym jest NLP?

- Po co nam to i dlaczego to ważne

2. Zadania NLP

- Tłumaczenie, klasyfikacja, chatboty i więcej

3. Krótka historia

- Od n-gramów i sieci rekurencyjnych do Transformerów

4. Tokenizacja i embeddingi

- Jak maszyny “widzą” tekst

5. Transformery

- Self-attention, BERT, GPT, encoder-decoder

6. Generowanie tekstu i LLM-y

- Od predykcji słów, do inteligentnych asystentów

7. Nowoczesne aplikacje

- RAG



Wstęp

Cyberpsychoza - gdy ślepo kopiujemy informacje, odpowiedzi z LLM bez rozumienia i weryfikacji.

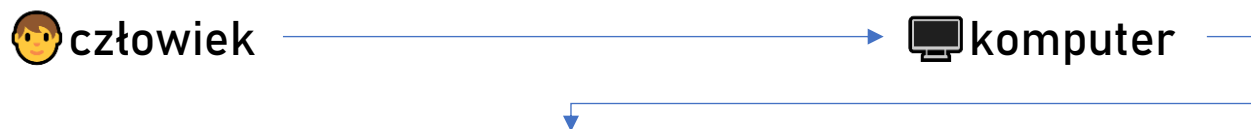


„inicjalizacja mózgu, milion neuronów, osiem mózgów!”



Czym jest NLP?

Przetwarzanie języka naturalnego – „interdyscyplinarna dziedzina, łącząca zagadnienia sztucznej inteligencji i językoznawstwa, zajmująca się automatyzacją analizy, rozumienia, tłumaczenia i generowania języka naturalnego przez komputer”

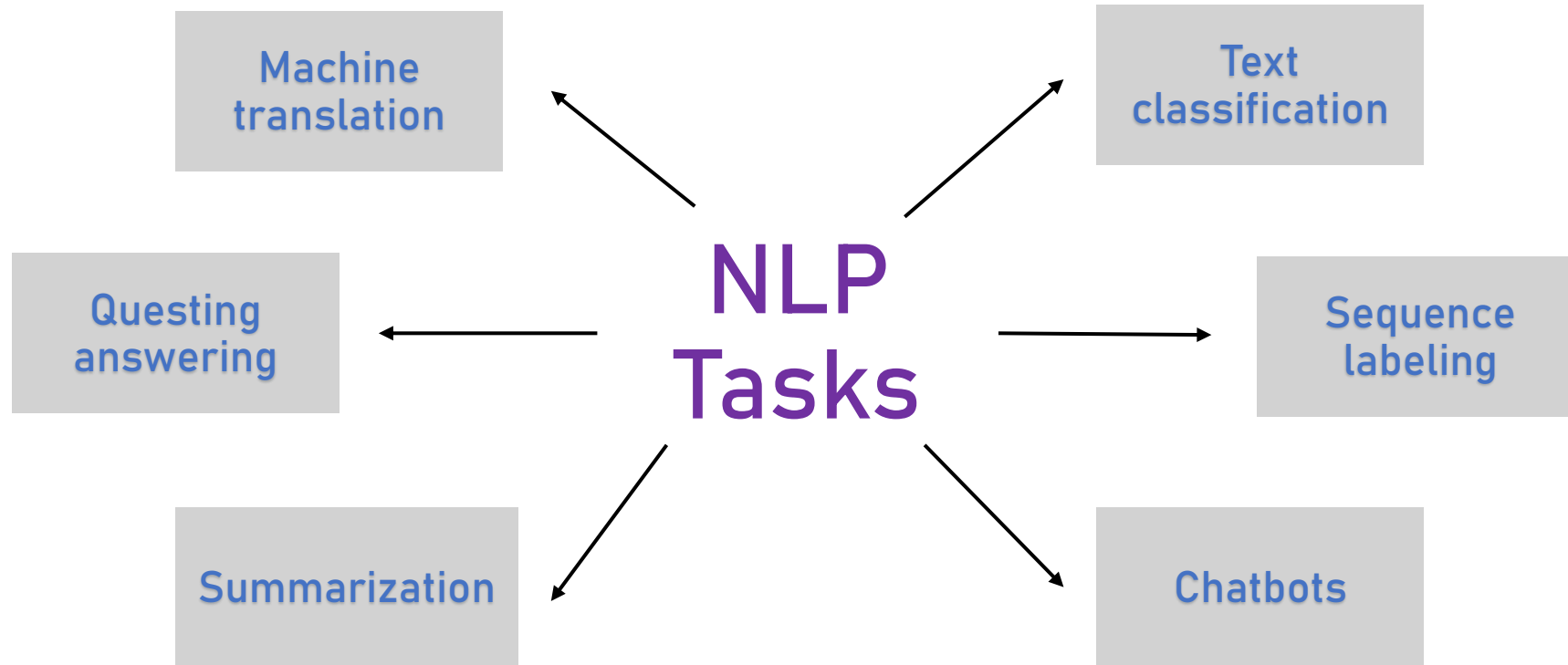


wynik:

- Chatboty 💬
- Tłumacze 🌐
- Asystenci głosowi 🎤
- Filtry spamu 📧
- Autokorekta ✎



Zadania NLP



Krótką historia

1. 1950 – początki

- Alan Turing -> test Turinga
- Georgetown-IBM (1954) -> pierwsze tłumaczenia maszynowe

2. 1960 – reguły

- ELIZA -> chatbot oparty na regułach (pattern matching, if, else)

3. 1980 – statystyka

- Modele probabilistyczne, n-gramy

4. 1990-2000

- Bag-of-Words, TF-IDF

5. 2000+ – deep learning

- Embeddingi, sieci rekurencyjne, transformery



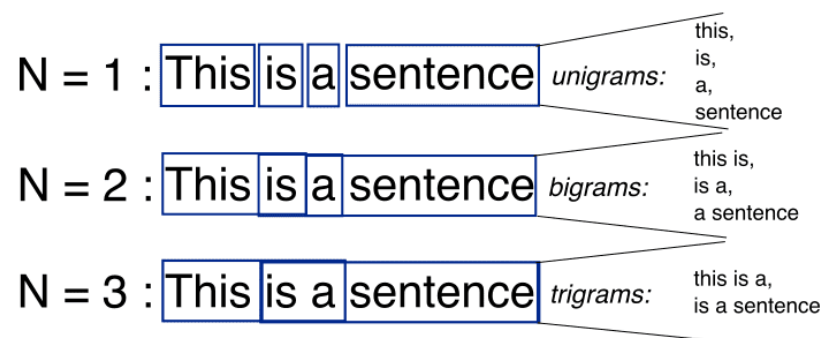
N-grams

N - gram – ciąg kolejnych słów, gdzie N to liczba słów na które chcemy patrzeć.

Bi -grams – para dwóch słów, które występują razem patrząc przed i po słowie.

Zliczanie częstotliwości w korpusie i wyciąganie proporcji -> zwykła statystyka. „Model” to tabela z prawdopodobieństwami

Pierwsze wersje Google Translate.





Zliczanie n-gramów:

- Heavy rain – 60 razy
- Heavy flood – 5 razy
- Heavy thunderstorm – 0 razy
- Etc.

Tabela prawdopodobieństw

$$P(\text{"There was heavy rain"}) = P(\text{"There", "was", "heavy", "rain"}) = P(\text{"There"}) P(\text{"was" | "There"}) \\ P(\text{"heavy" | "There was"}) P(\text{"rain" | "There was heavy"}).$$

Np. prawdopodobieństwo wystąpienia zdania „There was heavy rain” jest wyższe niż „There was heavy flood”, dlatego dla autouzupełniania następnym słowem po „heavy” będzie „rain”

$P(\text{rain}|\text{heavy}) \gg$ duża wartość
 $P(\text{flood}|\text{heavy}) \gg$ mała wartość



Dlaczego n-gramy to za mało?

- Tylko lokalny kontekst
 - Widzą tylko 1-2 słowa obok siebie
- Eksplozja kombinacji
 - Jeśli słownik to 50k słów to dla bi-gramów mamy 2,5mld możliwych par, nie mówiąc już o tri-gramach
- Brak uogólnienia
 - Gdy w korpusie nigdy nie wystąpiło pewne, słowo model n-gramowy nie będzie w stanie dać wyniku
- Brak prawdziwego rozumienia (statystyka)
 - N-gramy liczą tylko częstotliwości



Bag of Words

Zastosowanie:

Przede wszystkim klasyfikacja:

- Spam detection
- Analiza sentymentu
- Kategoryzacja dokumentów

Nowość względem n-gramów

- Prostsze i bardziej ogólne reprezentacje (brak eksplozji słownika)
- Skuteczne w połączeniu z algorytmami ML (np. SVM)

Ograniczenie:

- Brak kolejności i kontekstu -> nie nadaje się do zadań, które zależą od kolejności słów (tłumaczenie, rozpoznawanie mowy)
- Nie przewiduje niczego szczególnego, oprócz tego, że słowo występuje w zdaniu

„dog bites man” != „man bites dog”

Brak kolejności i kontekstu



Bag of Words – spam detection

Normal - "See you at the meeting tomorrow"

SPAM - „Hello John, you have won a free, click here”

↓ Kodujemy zdania w tablicę liczb

[0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1]

Każdy wektor był tak długi jak słownik

Vocabulary:

{Hello, free, you, have, won, see, you, money, meeting, tomorrow, click, here}

Length = 12

Possible labels:

SPAM

NOT SPAM

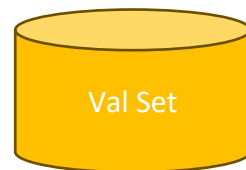
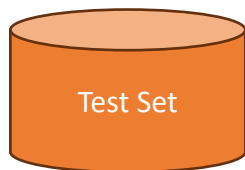
1/0

Nie ma to nic wspólnego z kolejnością wystąpienia słowa



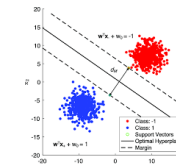
Bag of Words – spam detection

	labels	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...



Vectorization (Bag of words)

[0, 1, 0, 0, 1, ...]



Classifier (SVM, Naive Bayes etc.)

Output

NOT SPAM

SPAM



TF-IDF

Kluczowe różnice:

- Bag of words, dawał wektor binarny lub zliczenia (każde słowo liczyło się tak samo)
- TF-IDF wprowadza wagi

Dla każdego słowa w zdaniu/dokumencie obliczamy:

- TF – jak często dane słowo pojawia się w danym dokumencie
- IDF – jak rzadkie słowo jest w całym korpusie

Waga słowa (TF-IDF) = $TF \times IDF$

[0, 1, 0, 0, 1]

BoW (binarny)

[0.85, 0.90, 0.10, 0.05, 0.60]

TF-IDF

- Jeśli słowo jest częste w danym dokumencie, TF rośnie
- Jeśli słowo jest rzadkie, IDF rośnie
- Jeśli słowo jest wszędzie (np. the), IDF maleje -> jego waga ≈ 0

Ograniczenie:

- Nadal nie ma informacji o znaczeniu słowa i jego kolejności.

słowa, które są charakterystyczne i wyróżniają dokument, dostają największą wagę.



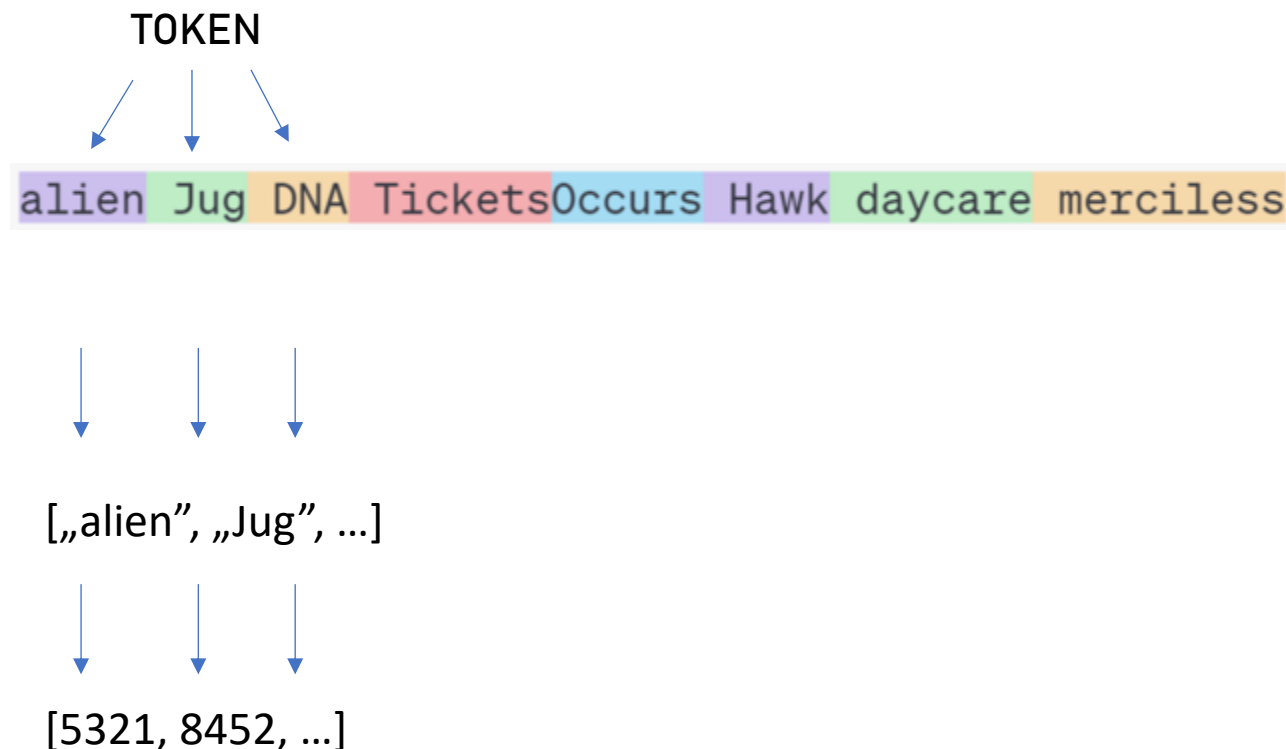
Tokenizacja

Co to jest?

- Podział tekstu na tokeny (najmniejsze jednostki, które model rozumie)
- Token \neq słowo \rightarrow to może być wyraz, interpunkcja, liczba, fragment wyrazu

Dlaczego takie ważne?

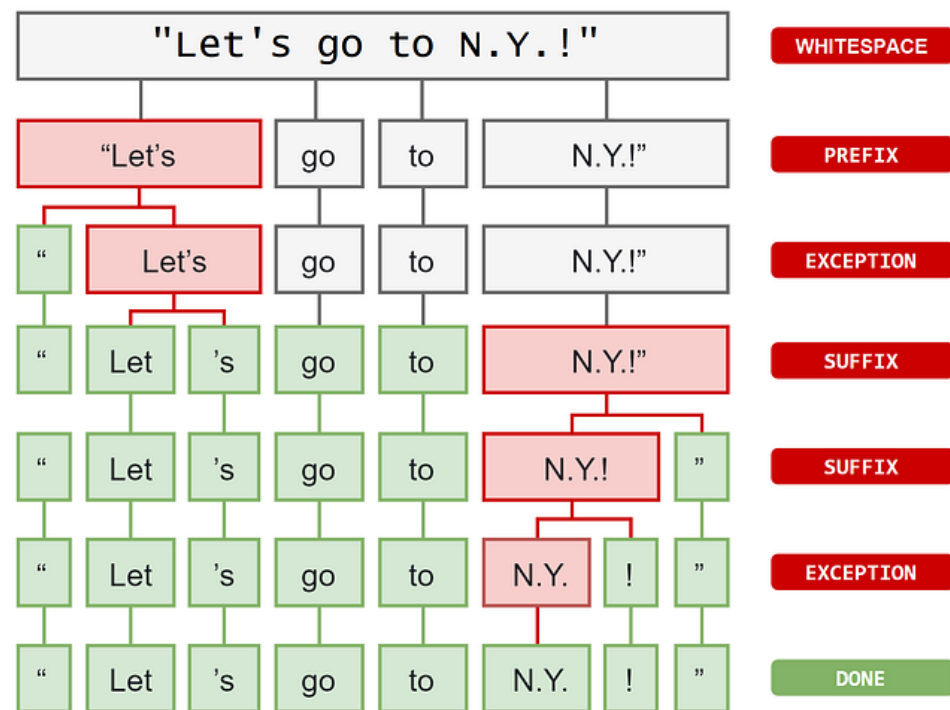
- Przygotowanie dla NLP
- Uproszczenie analizy



Tokenizacja

Wyzwania:

- Języki bez spacji (np. chiński)
- Wieloznaczność (np. *3.14* - liczba czy zdanie kropką?)
- Skróty i slang (np. „działa xd”)



Lematyzacja

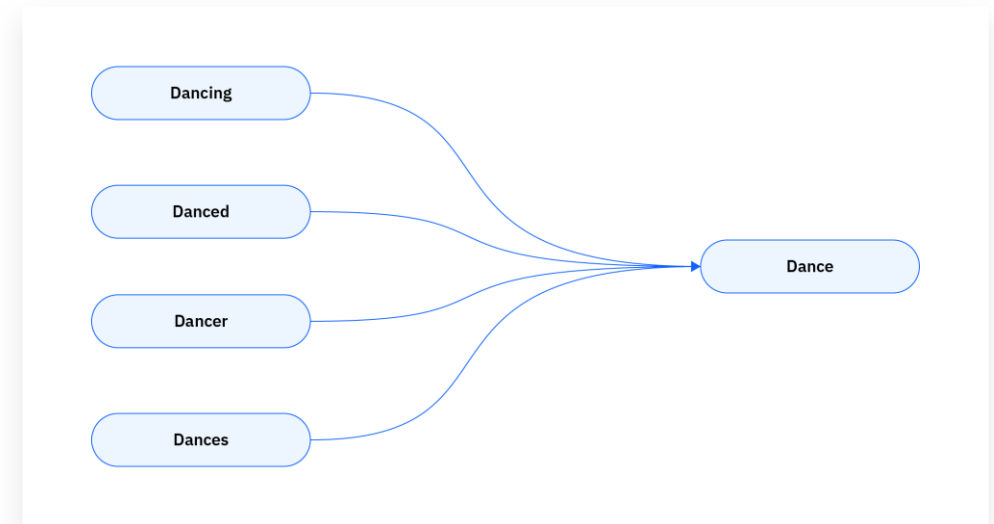
Co to jest?

- Proces sprowadzania słowa do jego formy podstawowej (lematu)

np. ładniejszy -> ładny, szedłem, idę -> iść, koty -> kot

Dlaczego to ważne?

- Ujednolicenie słów -> wszystkie odmiany traktujemy jako jedno słowo
- Lepsza analiza semantyki
- Redukcja wymiarów



Jak reprezentować tekst?

One-hot?

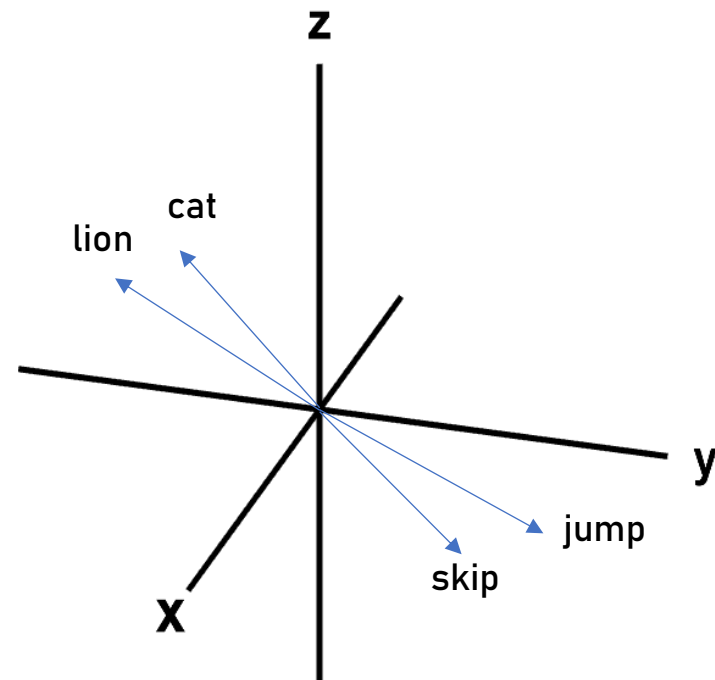
[0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1]

Raczej nie - nie nadaje się, dla zadań z kontekstem

Wykorzystujemy tokeny, które dostają wektor liczbowy opisujący jego znaczenie

Embedding - to reprezentacja słowa jako gęsty wektor w niskowymiarowej przestrzeni, w której podobne słowa są blisko siebie

Skąd je wziąć?



wektory semantyczne



Word2vec

World2Vec (2013, Mikolov)

- Metoda uczenia embeddingów
- Przekształca słowa w liczby, które można przetwarzać w NN
- Uchwytuje relacje semantyczne i składniowe
- Słowa o podobnym znaczeniu -> blisko w przestrzeni wektorowej

Dlaczego to było przełomowe?

- Zmieniło NLP: od one hot do embeddingów
- Umożliwiło skalowanie na miliardy słów
- Podstawa kolejnych modeli

Dwa podejścia

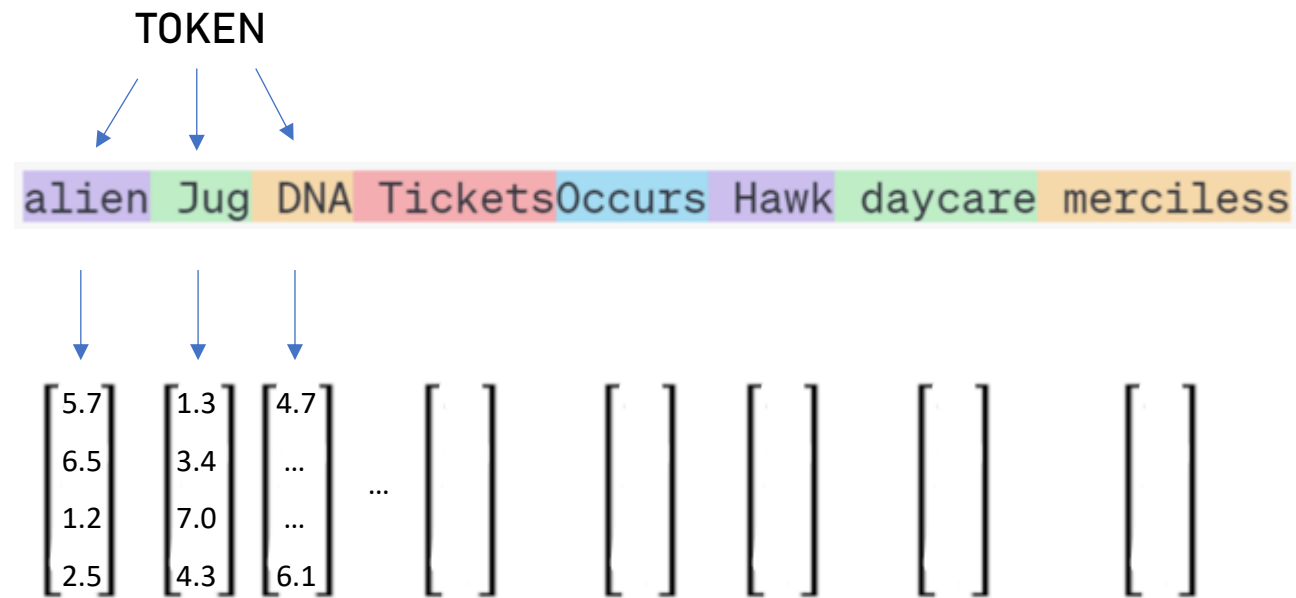
- CBOW – przewiduje słowo na podstawie kontekstu
- Skip-Gram – przewiduje kontekst na podstawie słowa

Ograniczenia:

- Embeddingi są statyczne – jedno słowo = jeden wektor



Embeddingi



RNN

Zanim nadeszła era transformerów, to właśnie RNN, a później LSTM i GRU, stanowiły podstawę przetwarzania języka naturalnego

- I ate
- I ate a sandwich
- I ate a sandwich with ham and cheese
- I ate a sandwich with ham and cheese because I was hungry after school

Sekwencje słów

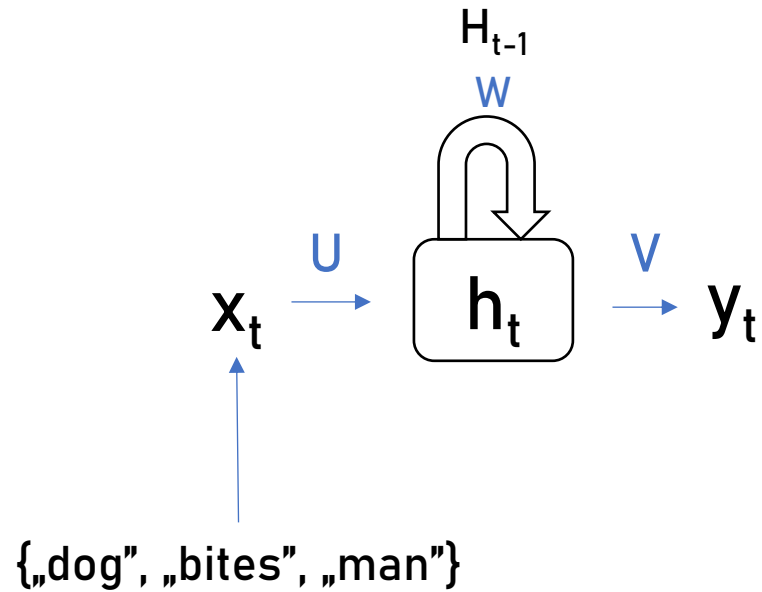
Mogą się różnić długością i wraz z jej wzrostem dają więcej kontekstu, ale standardowa sieć neuronowa widzi wejście jako płaski wektor

Co zrobić, aby rozróżnić
„dog bites man” i „man bites dog” ?

Możemy przetwarzać zdania jako sekwencje, zamiast „worków słów”



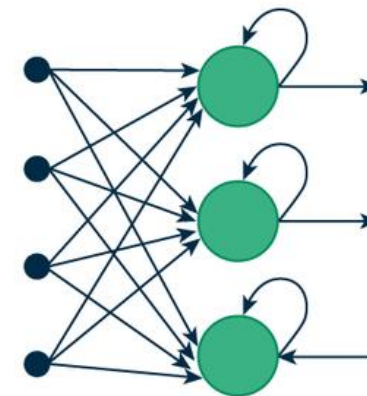
RNN



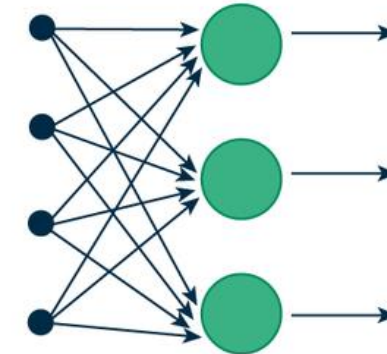
Połączenie rekurencyjne:

Ukryta warstwa przyjmuje dwa wejścia

1. Aktualne wejście x
2. Wyjście ukrytej warstwy z poprzedniego kroku



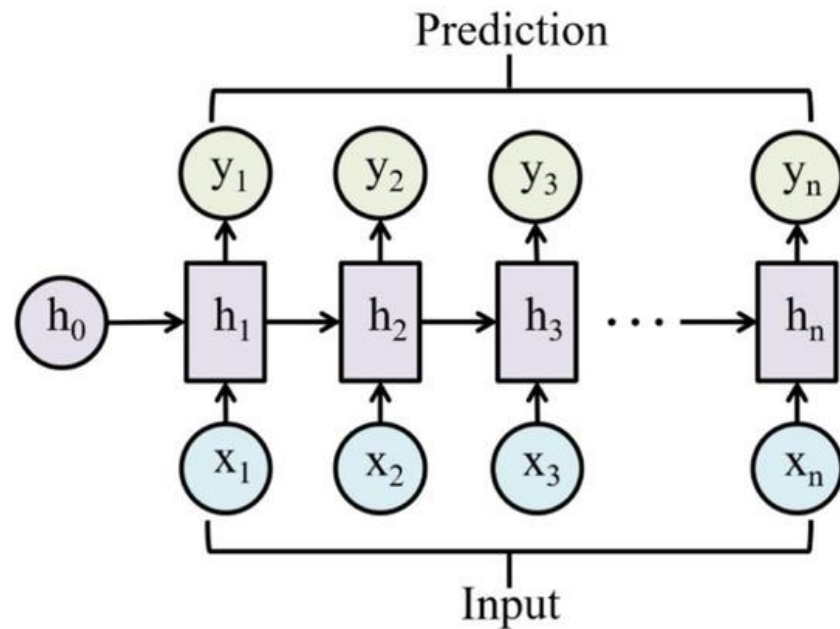
(a) Recurrent Neural Network



(b) Feed-Forward Neural Network



RNN



Trzy macierze wag:

- U – wagi między wejściem, a ukrytą warstwą
- W – wagi między poprzednim stanem ukrytym h_{t-1} , a aktualnym stanem h_t
- V – wagi między stanem ukrytym h_t , a wyjściem

$$h_t = \sigma(U_{xt} + Wh_{t-1} + b)$$

$$\hat{y}_t = g(Vh_t + c)$$

Gdzie:

σ – funkcja aktywacji w hidden layer

G – funkcja aktywacji w output layer

b – bias (przesunięcie) hidden state

c – bias (przesunięcie) output layer



RNN

Forward pass:

- RNN „unrolluje się” w czasie -> ta sama komórka dla każdego słowa
- Hidden state przekazuje informacje dalej
- Te same macierze U , W , V są używane na każdym kroku
- Dzięki temu model uczy się jednego sposobu przetwarzania sekwencji

Backpropagation:

- Gradienty liczone są przez wszystkie kroki w czasie.
- Ponieważ wagi są współdzielone, gradienty z poszczególnych kroków sumują się dla U , W , V
- Aktualizacja dotyczy jednego zestawu wag

Ograniczenia:

- Problem zanikającego gradientu (vanishing gradient)
- Pamięta tylko kilka kroków wstecz (lokalny kontekst)
- Exploding gradients (zbyt duże wartości -> niestabilny trening)
- Wolne uczenie i predykcja



RNN – LSTM/GRU

LSTM:

Dodaje mechanizm pamięci i bramki, które decydują

- Co zapamiętać
- Co zapominać
- Co wysyłać dalej

Efekt: potrafi uczyć się długich zależności

GRU:

Uproszcza LSTM przy zachowaniu zalet

Efekt: mniej parametrów, szybsze trenowanie

DLACZEGO TO WCIĄŻ ZA MAŁO?

- Przetwarzają słowa jeden po drugim – brak możliwości równoległa trenowania (wolne uczenie)
- Ograniczona pamięć
- Trudność w uczeniu złożonych zależności



Transformer

- Architektura wprowadzona w pracy „Attention Is All You Need” (Vaswani et al., 2017)
- Wykorzystuje mechanizm atencji
- Efektywne przetwarzanie danych
- Transformery obecnie dominują w wielu zadaniach NLP
- GPT – Generative Pre-Trained Transformer



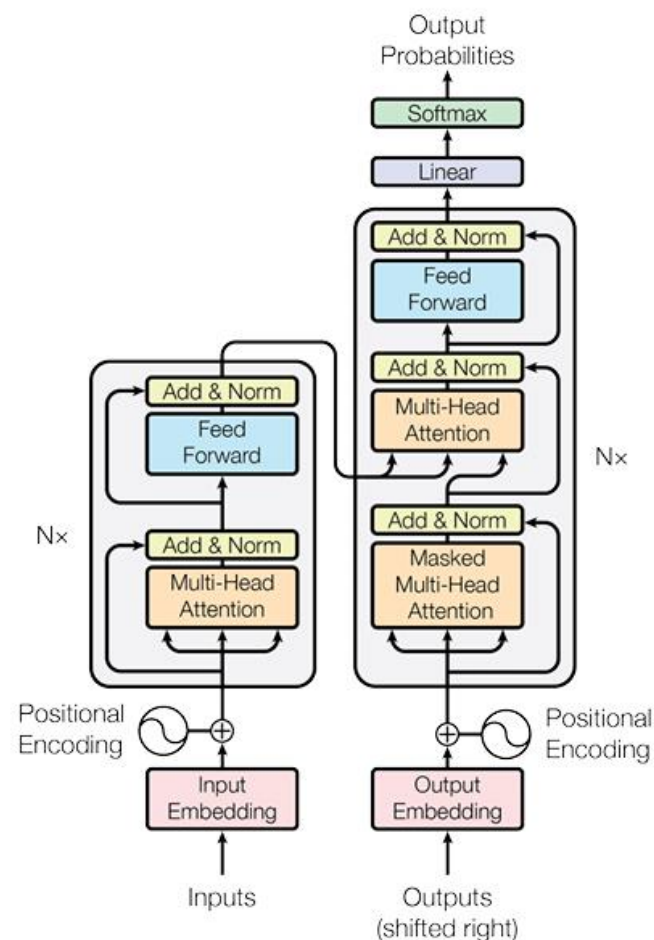
Transformer

Encoder-decoder:

- Encoder bierze sekwencję wejściową i buduje jej reprezentację
- Decoder generuje sekwencję wyjściową, używając reprezentacji z encodera + poprzednich tokenów

Kluczowe cechy:

- Self-Attention – każde słowo „waży” inne słowa w zdaniu
- Multi-Head Attention – wiele różnych perspektyw jednocześnie
- Positional Encoding



Transformer



BERT

- Tylko encoder
- Uczy reprezentacji
- Świetny do zadań rozumienia tekstu (klasyfikacja, QA)



GPT

- Tylko decoder
- Nastawiony na generowanie
- LLMs



Transformer

BERT (encoder)

I like to drink _____ in the morning

GPT (decoder)

Once upon a time there was a ...

W pełnym transformerze
Mamy Encoder i Decoder.

Po co nam sam Encoder?

- Rozumienie, a nie generacja tekstu
 - Analiza sentymentu
 - Ekstrakcja informacji
- Wskazywanie fragmentu odpowiedzi
- Wyjście stricte dopasowane do zadania

DO ZAPAMIĘTANIA

- Encoder – rozumienie (uczenie modelu wyciągania znaczenia z tekstu)
- Decoder – generacja tekstu (przewidujemy następny token) Tylko i wyłącznie prawdopodobieństwo następnego słowa



Transformer

Dlaczego Transformery wygrały?

- Równoległość (GPU friendly)
- Mogą uchwycić bardzo długi kontekst
- Skuteczność
- Uniwersalność, nie tylko NLP (np. ViT)



Transformer

Statyczne embeddingi (word2vec, GloVe)

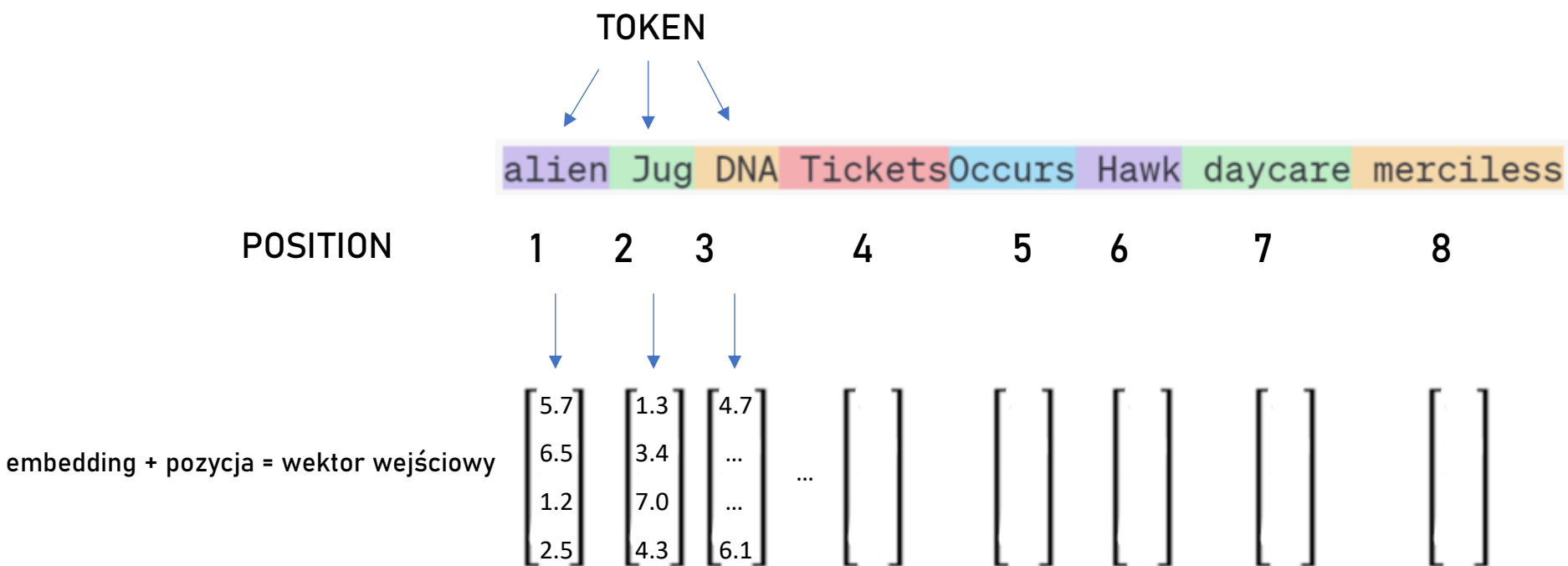
- Jedno słowo = jeden wektor niezależnie od kontekstu

Kontekstowe embeddingi (Transformer)

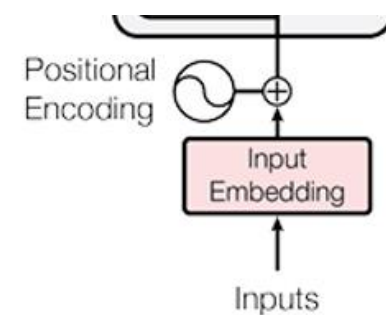
- Embedding słowa zależy od otoczenia
- „model” w „Machine learning model” \neq „model” w „Fashion model”
- Znaczenie „doprecyzowuje się” dzięki mechanizmowi **self-attention** -> wektory są aktualizowane podczas przetwarzania całej sekwencji



Transformer – Positional Encoding

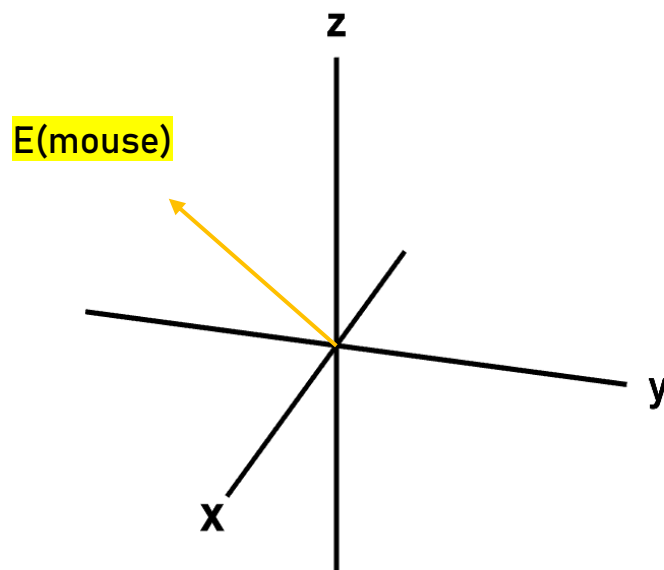
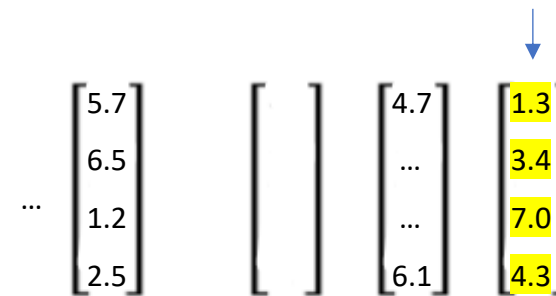
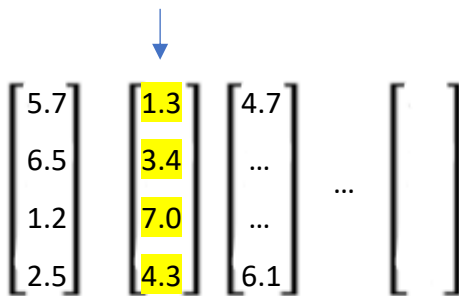
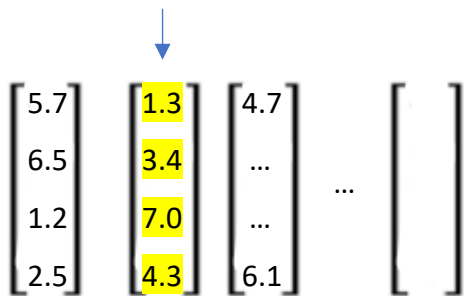


„pies goni kota, to nie to samo co kot goni psa”



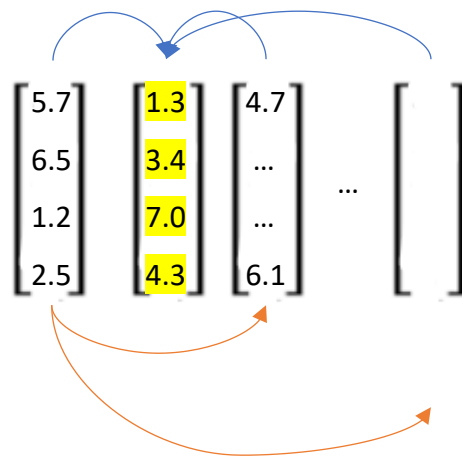
Transformer

The **mouse** ran across the kitchen floor. I need a new **mouse** for my computer Disney's most famous character is a **mouse**



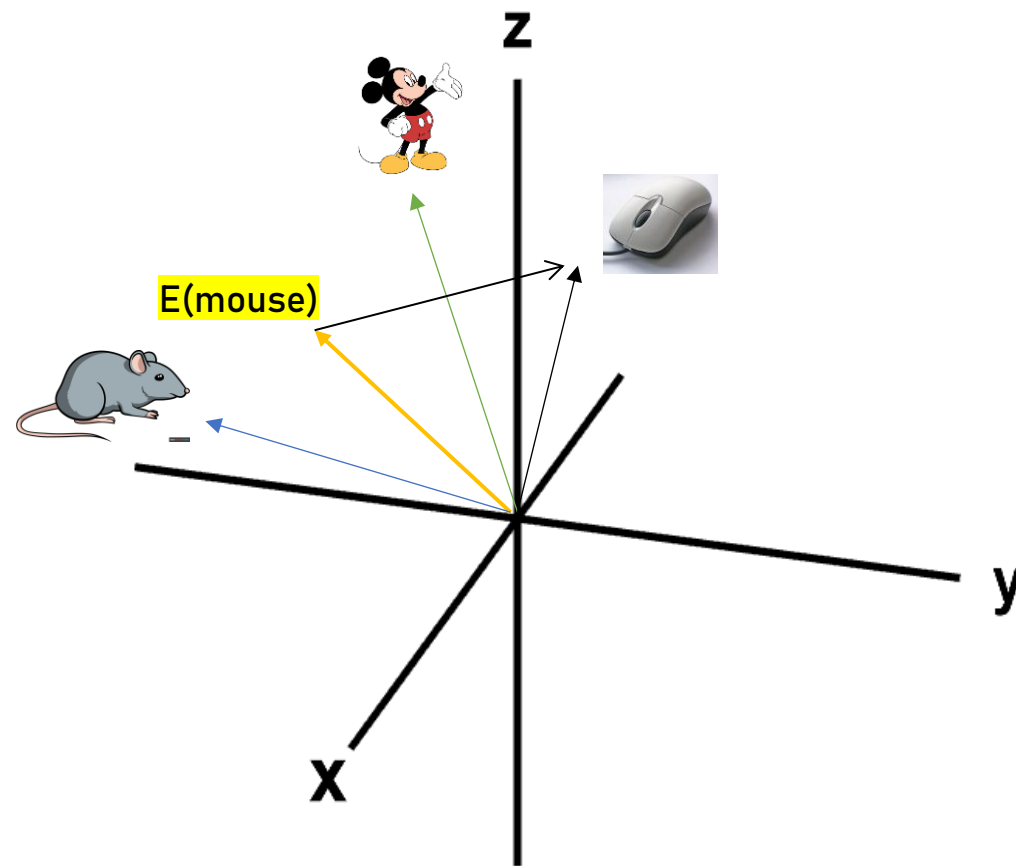
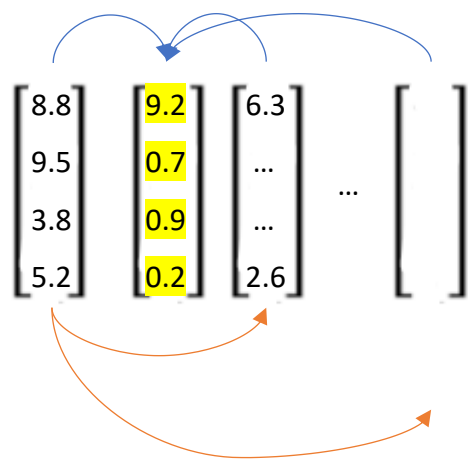
Transformer

I need a new mouse for my computer



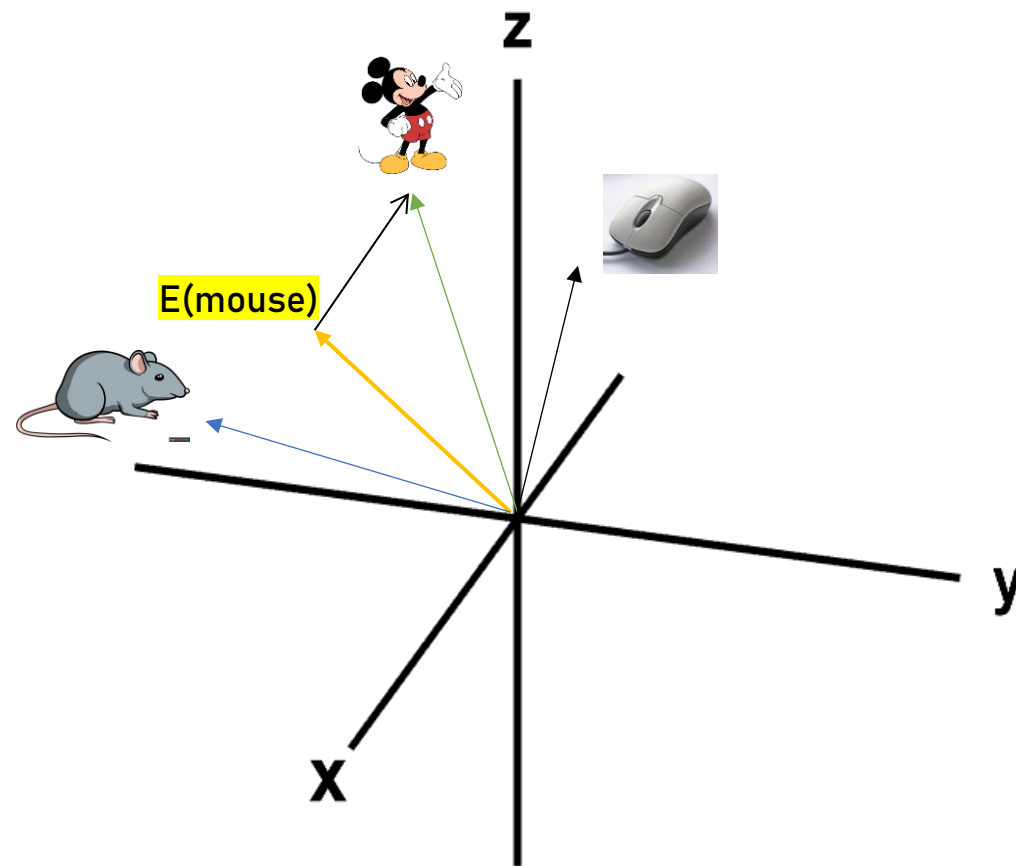
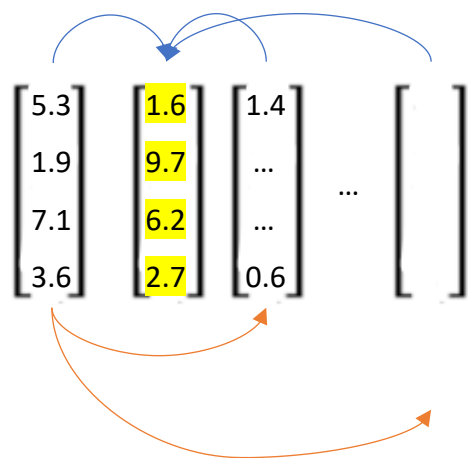
Transformer

I need a new mouse for my computer



Transformer

Disney's most famous character is a **mouse**



Transformer

Short distance

I need a new mouse for my computer



Long distance

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam interdum sapien ut tincidunt varius. Sed vitae ultrices nulla. Curabitur convallis mauris non lorem scelerisque, ut tincidunt elit vehicula. Cras aliquet luctus orci, a dictum lectus. Suspendisse potenti. Pellentesque accumsan ligula vel elit convallis, nec sagittis purus posuere. Integer ut velit quis leo dapibus pulvinar. Donec sit amet fringilla lorem. Morbi tempor nisl sed metus feugiat, in feugiat mauris congue. Vivamus accumsan neque nec lectus vestibulum, non hendrerit nisl blandit. Proin cursus sapien ac sem ornare, a luctus ipsum fermentum. Etiam rhoncus nisl vitae venenatis varius. Sed quis sem eu mauris pretium fermentum. Nunc imperdiet purus in magna volutpat, vitae cursus libero malesuada.



Transformer

Każdy token zamieniamy na trzy wektory:

- Query (Q) – czego szukam?
- Key (K) – jakie mam informacje?
- Value (V) – treść informacji, którą mogę przekazać

Działanie:

1. Porównuje Query jednego słowa z Key innych słów -> podobieństwo (waga uwagi)
2. Wagi normalizowane są softmaxem
3. Obliczanie ważonej sumy Value innych słów -> nowy embedding słowa

Efekt: każde słowo może „zapytać” inne słowa, na ile są dla niego ważne.



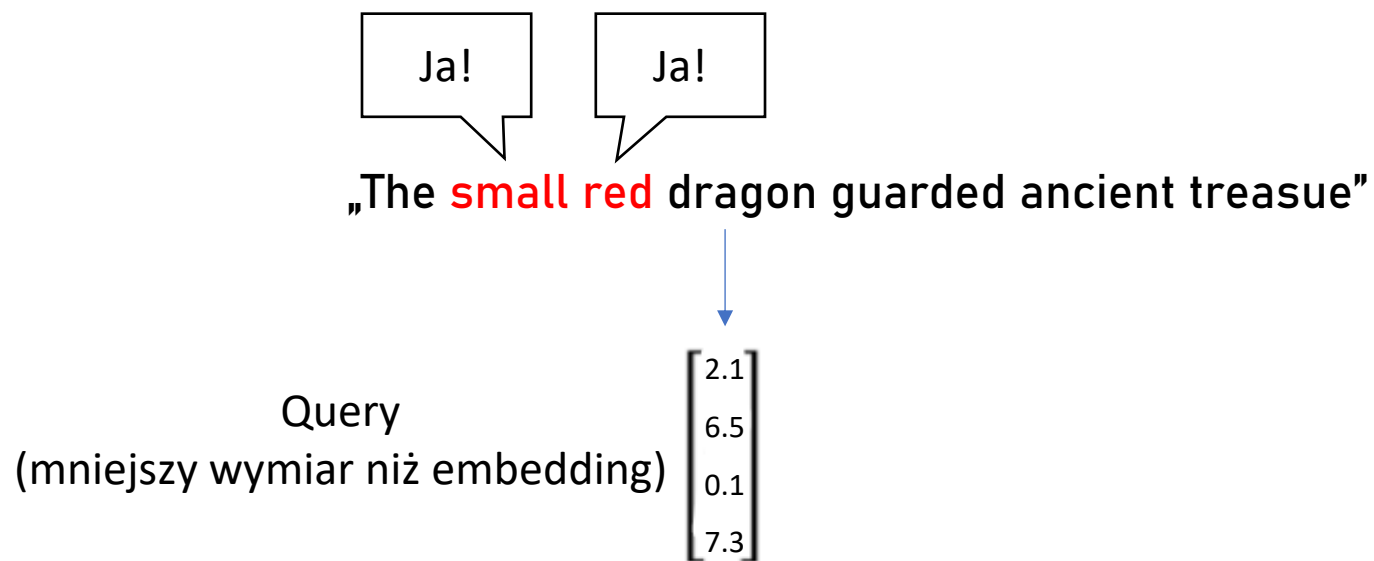
Transformer – mechanizm atencji

Czy są jakieś
przymiotniki
przede mną?

„The small red **dragon** guarded ancient treasure”



Transformer – mechanizm atencji



Transformer – mechanizm atencji

Ale skąd ten wektor Query?

$$\mathbf{W} = \begin{matrix} & \mathbf{W}_Q \\ \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix} \end{matrix} \times \begin{bmatrix} 5.3 \\ 1.9 \\ 7.1 \\ 3.6 \end{bmatrix} = \vec{Q}$$

Robimy tak dla każdego embeddingu. Każdy embedding („słowo”) generuje wektor query (w uproszczeniu pytanie)



Transformer – mechanizm atencji

W tym samym czasie, na tej samej zasadzie mamy wektor Key.

$$\mathbf{W} = \begin{matrix} & \mathbf{W}_K \\ \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix} \end{matrix} \times \begin{bmatrix} 7.3 \\ 5.2 \\ 3.0 \\ 3.1 \end{bmatrix} = \vec{K}$$



Transformer – mechanizm atencji

		small	red	dragon
		Q_1	Q_2	Q_3
small	K_1	$K_1 \times Q_1$	$K_1 \times Q_2$	$K_1 \times Q_3$
red	K_2	$K_2 \times Q_1$	$K_2 \times Q_2$	$K_2 \times Q_3$
dragon	K_3	$K_3 \times Q_1$	$K_3 \times Q_2$	$K_3 \times Q_3$



Transformer – mechanizm atencji

		small	red	dragon
		Q_1	Q_2	Q_3
small	K_1	$K_1 \times Q_1$	$K_1 \times Q_2$	$K_1 \times Q_3$
red	K_2	$K_2 \times Q_1$	$K_2 \times Q_2$	$K_2 \times Q_3$
dragon	K_3	$K_3 \times Q_1$	$K_3 \times Q_2$	$K_3 \times Q_3$



Transformer – mechanizm atencji

		small	red	dragon
		Q_1	Q_2	Q_3
small	K_1	-56.4	3.2	92.3
red	K_2	-45.2	-23.3	91.4
dragon	K_3	-22.1	12.9	-3.1



Transformer – mechanizm atencji

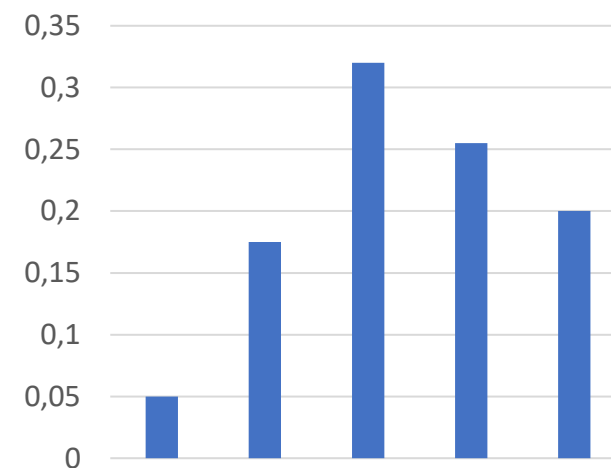
-56.4	3.2	92.3
-45.2	-23.3	91.4
-22.1	12.9	-3.1
...
...



softmax



$$0.05 + 0.175 + 0.320 + 0.255 + 0.200 = 1$$



Transformer – mechanizm atencji

...	...	0.58
...	...	0.42
...	...	0.0
...
...

Attention pattern

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



Transformer – mechanizm atencji

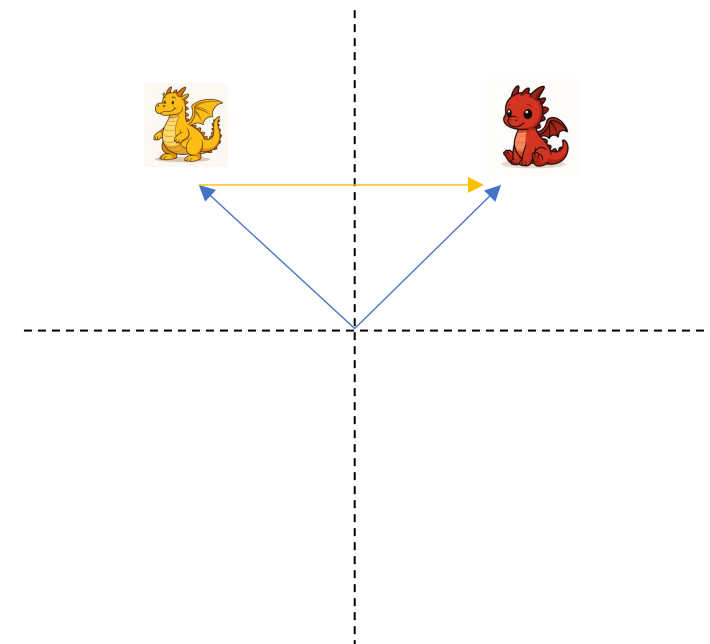
No dobra, mamy Q, mamy K, mamy softmax, co dalej?

Mamy trzeci wektor V, które posiada informację jaką ma przekazać

$$W_V = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \dots & w_{nn} \end{bmatrix} \times \begin{bmatrix} 5.3 \\ 1.9 \\ 7.1 \\ 3.6 \end{bmatrix} = \vec{V}$$

V może być już innego rozmiaru niż Q i K

$$Attention(Q, K, V) = softmax\left(\frac{K^T Q}{\sqrt{d_k}}\right) V$$



Hmm.. Dragon patrzy mocno na Key „red” (waga 0.6) i trochę na small (waga 0.3) -> nowy embedding dragon = $0.6 \times V_{red} + 0.3 \times V_{small}$



Transformer – mechanizm atencji

...	...	$0.58 \times \vec{V}_1$
...	...	$0.42 \times \vec{V}_2$
...	...	$0.0 \times \vec{V}_3$
...
...

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



„The small red **dragon** guarded ancient **treasure**”



„The **small** **red** **dragon** guarded **ancient** **treasure**”





Transformer – mehanizem atenciji

\vec{E}_3

...	...	$0.58 \times \vec{V}_1$
...	...	$0.42 \times \vec{V}_2$
...	...	$0.0 \times \vec{V}_3$
...
...

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

 $\vec{E}_3 + \Delta \vec{E}_3 = \vec{E}_3$ 



Transformer

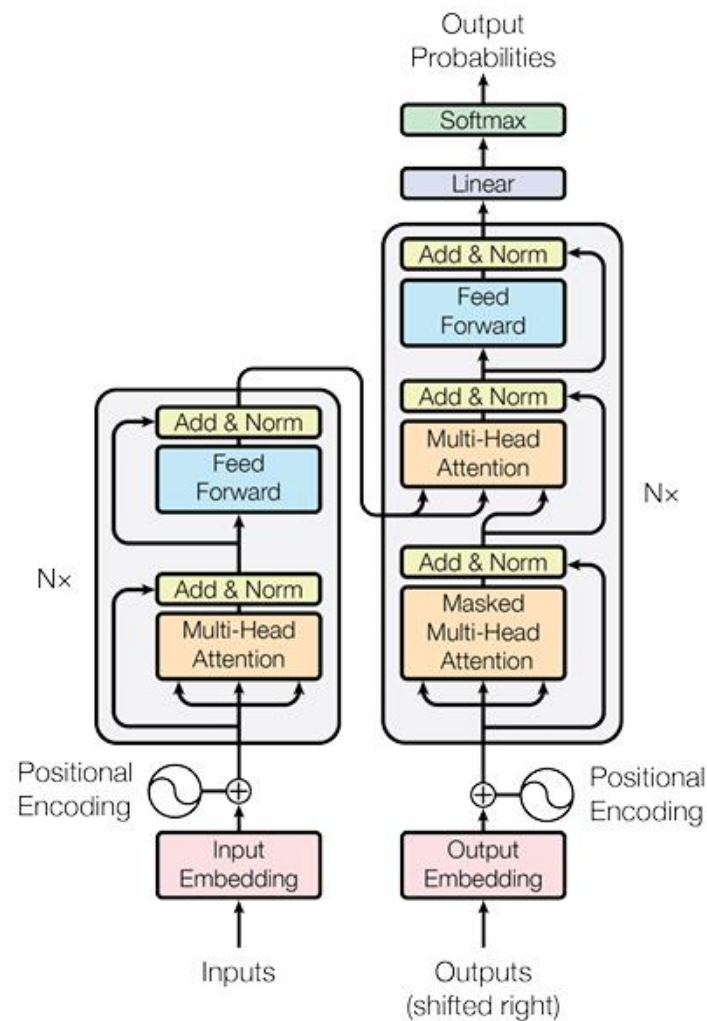
Multi Head Attention

- Model dzięki temu lepiej rozumie złożone relacje
- Każda uczy się patrzeć na tekst z innej perspektywy



np.

- Przymiotniki \leftrightarrow rzeczowniki (**small** \rightarrow **dragon**)
- Czasownik \leftrightarrow podmiot (**dog** \rightarrow **barks**)
- Jednostki czasu (**yesterday** \leftrightarrow **tomorrow** \leftrightarrow **today**)
- Słowa w tej samej domenie (**king** \leftrightarrow **queen**)
- I wiele innych

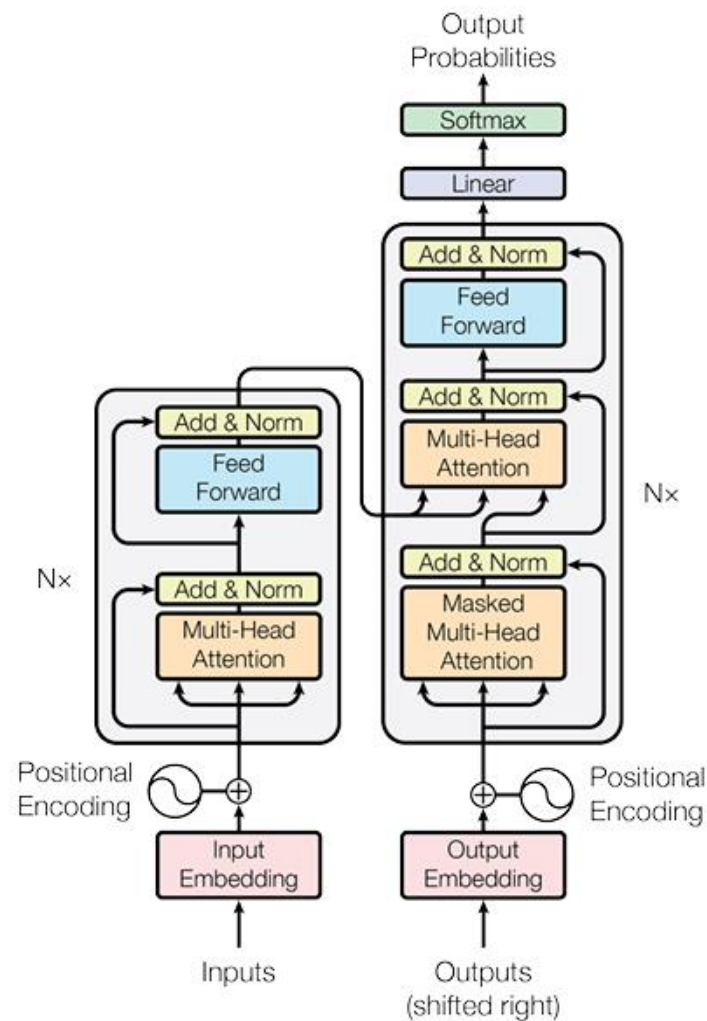


Transformer

Architektura raz jeszcze (Encoder)

1. Multi Head Attention
2. Połączenia rezydualne + normalizacja
3. Feed-Forward Netowrk
4. Znowu połączenie rezydualne + normalizacja

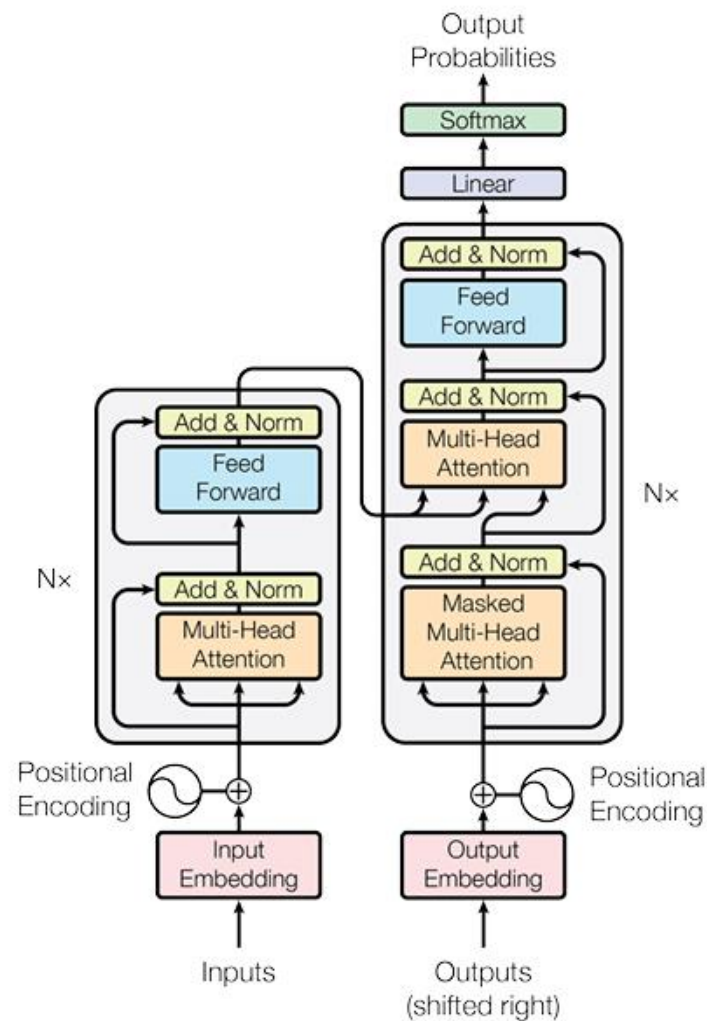
W BERT, wynikiem jest często embedding [CLS]
używany jako reprezentacja całego zdania



Transformer

Architektura raz jeszcze (Encoder-decoder)

1. Encoder (całe zdanie wejściowe) i output jako sekwencja kontekstowych embeddingów
 2. Decoder – Masked Self Attention (następny slajd)
 3. Encoder-Decoder Attention -> Query z dekodera patrzą na Key/Value z encodera (łączenie kontekstów)
- T5 (encoder-decoder) – wynikiem jest np. wygenerowana sekwencja tekstu (np. tłumaczenie)
- ALB0
- Audio (encoder) i transkrypcja (decoder) jako wynik



Transformer – mechanizm uwagi

maskowanie ->

+3.21	some val	some val	some val	some val	some val
$-\infty$	-0.12	some val	some val	some val	some Val
$-\infty$	$-\infty$	+0.89	some val	some val	some Val
$-\infty$	$-\infty$	$-\infty$	+5.12	some Val	some Val
$-\infty$	$-\infty$	$-\infty$	$-\infty$	+2.91	some Val
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	+0.67

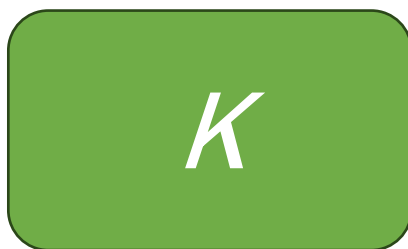
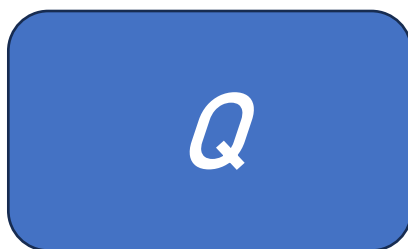
softmax ->

1.00	some val	some val	some val	some val	some val
0	0.25	some val	some val	some val	some Val
0	0	0.48	some val	some val	some Val
0	0	0	0.46	some Val	some Val
0	0	0	0	0.52	some Val
0	0	0	0	0	0.78

Decoder



Transformer – mechanizm atencji



One Head of Attention

Cały proces jest parametryzowany przez 3 odrębne macierze
Wszystkie wypełnione parametrami, które się zmieniają w trakcie uczenia

Softmax

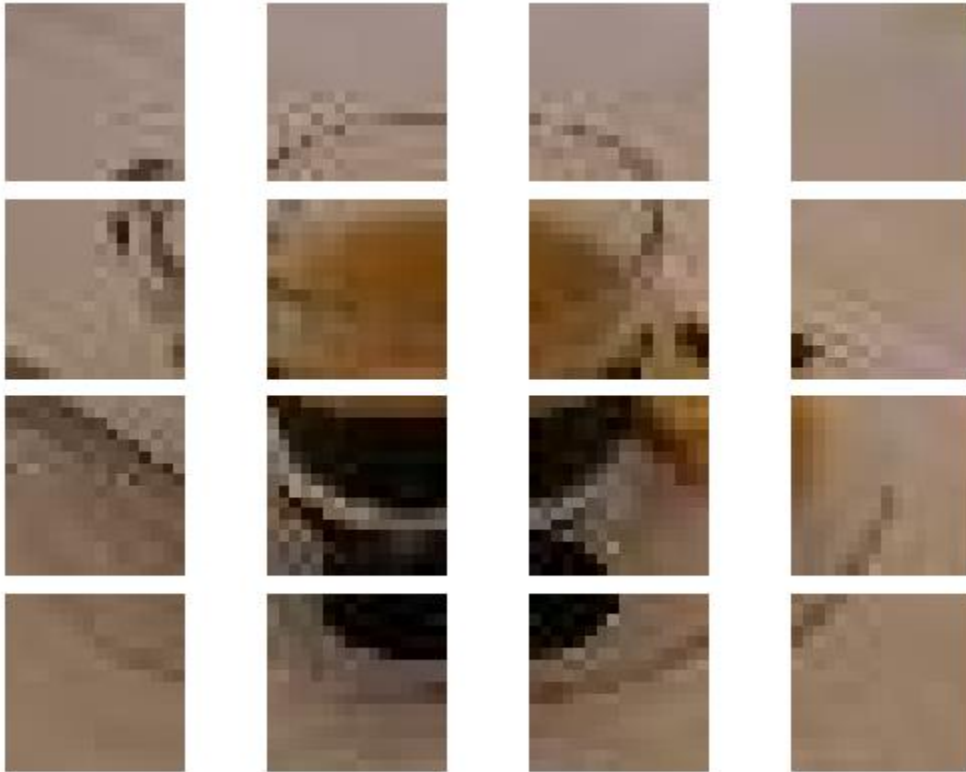
$$\left[\frac{\begin{array}{c|c} \text{Query Matrix} & \times & \text{Transposed Key Matrix} \\ \hline \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline & & & & \\ \hline & & & & \\ \hline & & & & \\ \hline \end{array} & & \begin{array}{|c|c|c|c|c|} \hline & & & & \\ \hline & & & & \\ \hline & & & & \\ \hline & & & & \\ \hline \end{array} & \\ \hline \sqrt{d} & = & \begin{array}{c} \text{Unnormalized} \\ \text{Attention Scores} \end{array} & \rightarrow & \begin{array}{c} \text{Masked} \\ \text{Attention Scores} \end{array} \end{array} \right]$$

Unnormalized Attention Scores				
1.1	2.3	5.3	2.1	-1.2
-0.1	0.5	1.3	-0.1	6.1
2.3	0.2	3.3	-1.0	-0.4
0.3	1.2	-0.3	5.0	1.4
5.1	-2.9	-1.1	-4.2	0.4

Masked Attention Scores				
1.1	-∞	-∞	-∞	-∞
-0.1	0.5	-∞	-∞	-∞
2.3	0.2	3.3	-∞	-∞
0.3	1.2	-0.3	5.0	-∞
5.1	-2.9	-1.1	-4.2	0.4



Ciekawostka



- Tokeny mogą być fragmentami zdjęcia (patchami)
- Każdy patch staje się analogią „słowa” w zdaniu, a cały obraz zmienia się w sekwencję „wizualnych tokenów”



LLM

Czym są?

- Ogromne modele Transformerowe (setki miliardów/może więcej parametrów)
- Trenowane na miliardach danych
- Dzięki self-attention potrafią „rozumieć” kontekst i tworzyć spójne odpowiedzi



Multi Head Attention

- W pojedynczej głowie model patrzy na relacje w jednej „podprzestrzeni” embeddingów
- Multi head = wiele równoległych głów, każda z innymi macierzami wag
- Każda może uchwycić inne zależności
- Bogata z wieloma perspektywami reprezentacja



LLM i Transformer

Unembedding (rzutowanie na słownik)

1. Mamy naszą macierz embeddingów (np. 1 wiersz jeden embedding, a kolumna to jego długość)
2. Dekoder produkuje kontekstowe wektory
3. Każdy taki embedding/wektor (na końcu) mnożymy przez macierz unembeddingu (transponowana embeddingu)
4. Wynik to logity, surowe wartości dla wszystkich słów w słowniku
5. Logity puszczaemy przez softmax (czasem dzieląc przez temperature), żeby dostać rozkład prawdopodobieństwa.

$$W_E = \begin{array}{c|ccc} & \begin{array}{c} 0.1 \\ \text{Token0} \end{array} & 0.5 & -0.2 & 0.7 \\ & \begin{array}{c} -0.3 \\ \text{Token1} \end{array} & 0.8 & 0.0 & 0.4 \\ & \begin{array}{c} 0.9 \\ \text{Token2} \end{array} & -0.1 & 0.2 & -0.5 \end{array}$$



$$W_E^T = \begin{array}{ccc|c} 0.1 & -0.3 & 0.9 & \\ 0.5 & 0.8 & -0.1 & \\ -0.2 & 0.0 & 0.2 & \\ 0.7 & 0.4 & -0.5 & \end{array}$$

Słowa -> tokeny -> embeddingi -> macierz, w której są nasze embeddingi E1, E2 ..
(Rozmiar – ilość tokenów x wymiar embeddingu)



LLM i Transformer

Unembedding (rzutowanie na słownik)

1. Mamy naszą macierz embeddingów (np. 1 wiersz jeden embedding, a kolumna to jego długość)
2. Dekoder produkuje kontekstowe wektory
3. Każdy taki embedding/wektor (na końcu) mnożymy przez macierz unembeddingu (transponowana embeddingu)
4. Wynik to logity, surowe wartości dla wszystkich słów w słowniku
5. Wiersze (z logitami) puszczamy przez softmax (czasem dzieląc przez temperaturę), żeby dostać rozkład prawdopodobieństwa.

wiersz logitów = $(1.0, -0.5, 0.3, 0.7)$ \times

0.1	-0.3	0.9
0.5	0.8	-0.1
-0.2	0.0	0.2
0.7	0.4	-0.5



jeden wiersz = $(0.7, 0.23, 0.36)$

Wektor długości 3 (bo mamy 3 tokeny w słowniku)
Jeśli słownik = 50000 tokenów, to JEDEN wektor jest takiego wymiaru

Kolumny odpowiadają konkretnym tokenom w słowniku

Słowa -> tokeny -> embeddingi -> macierz, w której są nasze embeddingi E1, E2 ..
(Rozmiar – ilość tokenów x wymiar embeddingu)



LLM i Transformer

Softmax

- Softmax bierze całą wiersz (wszystkie logity dla jednej pozycji) i normalizuje je tak, żeby zamieniły się w rozkład prawdopodobieństwa

→ [0.25 0.03 0.55 0.17]



Największe prawdopodobieństwo, czyli wybieramy token o ID=2
(np. „dragon”)

TRENING VS GENEROWANIE

- | | |
|---|--|
| 1. Dla każdego tokenu wyjściowego przewidujemy następny token | 1. Patrzymy tylko na ostatni wektor (ostatnie „słowo” w sekwencji) |
| 2. Każdy wektor jest używany równolegle do przewidywać | 2. Działamy autoregresyjnie |



LLM i Transformer

Temperatura – hiperparamter

- $T = 1$ -> zwykły softmax $\longrightarrow [0.25 \quad 0.03 \quad 0.55 \quad 0.17]$
- $T < 1$ (np. 0.5) -> rozkład robi się bardziej ostry, największe prawdopodobieństwo rośnie, reszta maleje $\longrightarrow [0.10 \quad 0.01 \quad 0.85 \quad 0.04]$
- $T > 1$ (np. 2.0) -> rozkład spłaszcza się, mniejsze prawdopodobieństwa dostają więcej szans (RYZYKO NONSENSU) $\longrightarrow [0.22 \quad 0.09 \quad 0.38 \quad 0.31]$



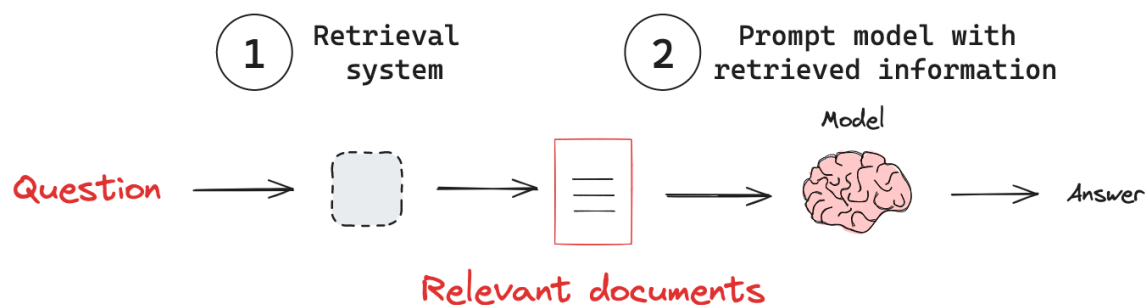
Nowoczesne aplikacje - RAG

RAG (Retrieval-Augmented Generation)

- RAG = połączenie LLM + wyszukiwania informacji (retrieval)
- Model nie polega wyłącznie na tym co „ma w parametrach”

- Zmniejszona halucynacja
- Dostęp do konkretnych informacji
- Personalizacja pod konkretne dane

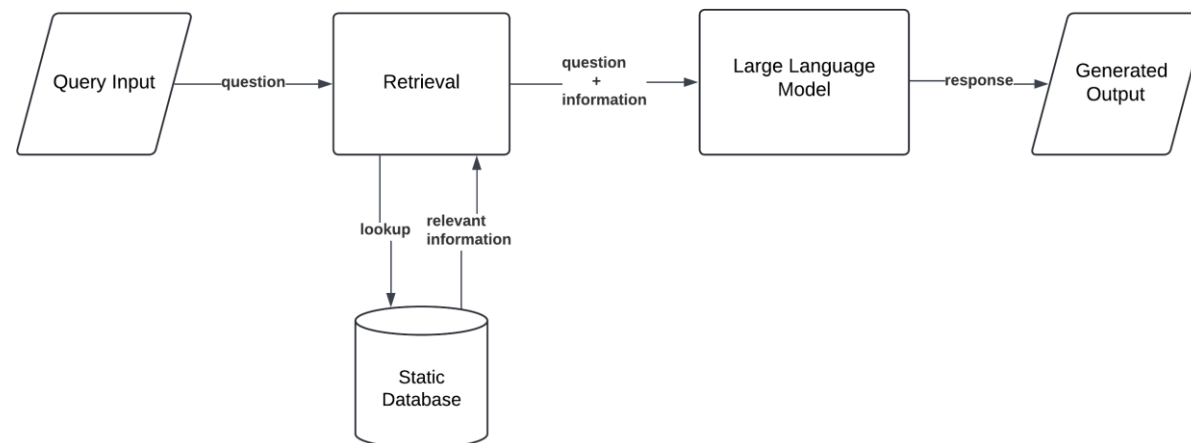
LLM + baza wiedzy = RAG



Nowoczesne aplikacje - RAG

1. Zapytanie użytkownika (prompt)
2. Retrieval – wyszukiwarka wektorowa (np. FAISS)
3. Augmentacja promptu
4. Generowanie odpowiedzi (LLM)

Użytkownik -> Wektorowa baza -> LLM -> Odpowiedź



Nowoczesne aplikacje - RAG

Źródła danych:

- Dokumenty tekstowe (PDF, json, html etc.)
- Strony internetowe / artykuły
- Dane z jakiejś domeny np. prawo
- Dane firmowe – dokumentacje etc.
- Bazy danych
- Systemy zewnętrzne API
- Grafy wiedzy





Dziękujemy za uwagę!

Jakub Gilewicz | Jakub Buszyński