Podstawy Sztucznej Inteligencji (PSI)

Lab 4 - omówienie

Pytania? Wątpliwości? Uwagi?

Częste kwestie

Natural language processing (NLP)

Wstęp

Zadania w NLP

- klasyfikacja / regresja tekstów, np. sentiment analysis, CTR prediction
- klasyfikacja tokenów, np. part-of-speech (POS) tagging, Named Entity Recognition (NER)
- machine translation
- text summarization
- topic modelling
- question answering (QA): extractive, generative
- text generation / prompt completion
- semantic search

Tokeny

- **token** - sformalizowane pojęcie "słowa"
- może być:
  - całym słowem lub **podsłowem (subword)**
  - jego **lematem (lemma)** lub **rdzeniem (stem)**
  - inną jednostką, np. liczbą, emoji etc.
  - interpunkcją
- podział tekstu na tokeny to **tokenizacja (tokenization)**

```
He remains characteristically
confident and optimistic.

[
  'He',
  'remains',
  'characteristic',
  '##ally',
  'confident',
  'and',
  'optimistic',
  '.'
]
```

Stemming vs Lemmatization

Podejście klasyczne

- tokenizacja całych słów + stemming / lematyzacja
- usuwamy **stop words** - popularne, mało znaczące słowa, np. "the", "he", "is"
- budujemy słownik **bag-of-words (BoW)**, zawierający otrzymane **słownictwo (vocabulary)**
  - wektor częstotliwości poszczególnych tokenów
  - ignoruje kolejność, mapowanie token -> pozycja jest zwykłym haszem
- **zastosowanie:** tam, gdzie liczą się same słowa, a nie złożona semantyka i kolejność
- np. topic modelling, prosta klasyfikacja tekstu (filtry spamu)

Wektory słów i sieci RNN

Wektoryzacja tokenów

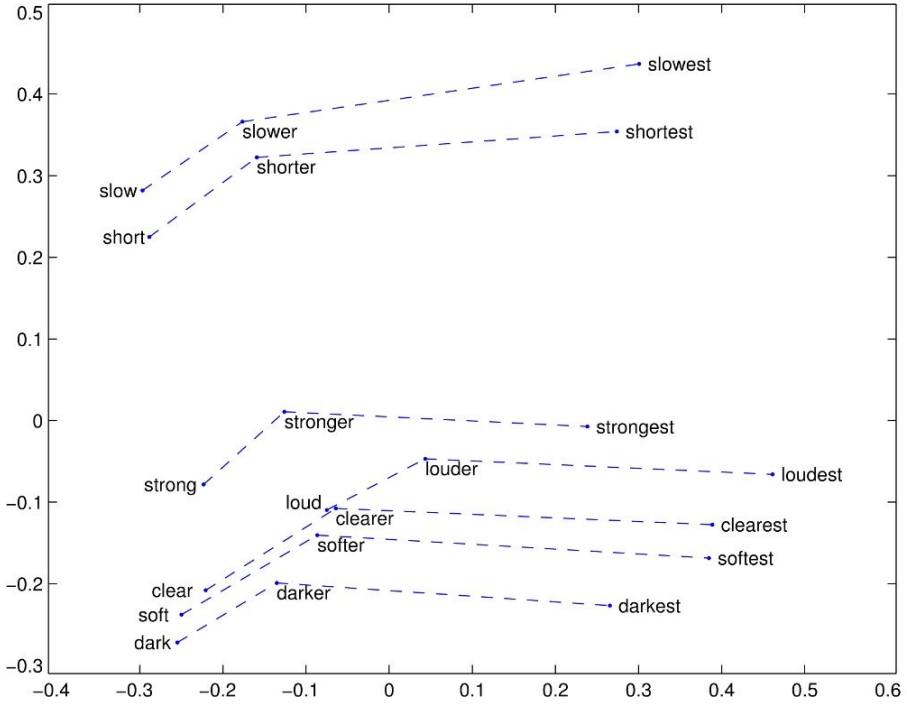
- **idea:** skoro CNNy dobrze wektoryzowały obrazy, to może dałoby się to zrobić z tokenami?
- **algorytmy:** word2vec, GloVe (Global Vectors for word representation)
- **word embeddings** - reprezentacje słów za pomocą wektorów w przestrzeni ciągłej, zachowujące sens semantyczny
- te algorytmy są **bezkontekstowe** - słowo "zamek" ma zawsze taki sam embedding, niezależnie od zdania (kontekstu)

Wektoryzacja tokenów - sens semantyczny

word2vec

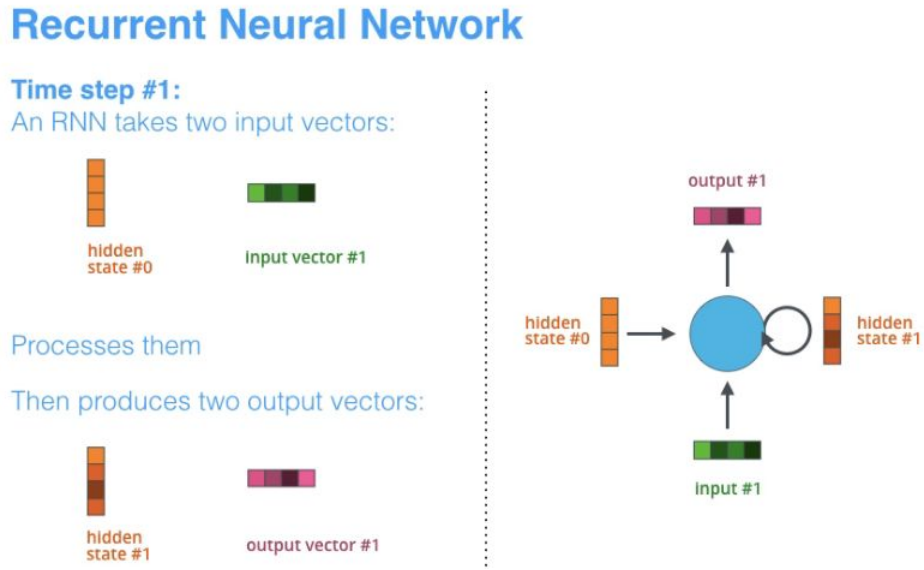
`King - Man + Woman = Queen`

Po prawej - **GloVe**

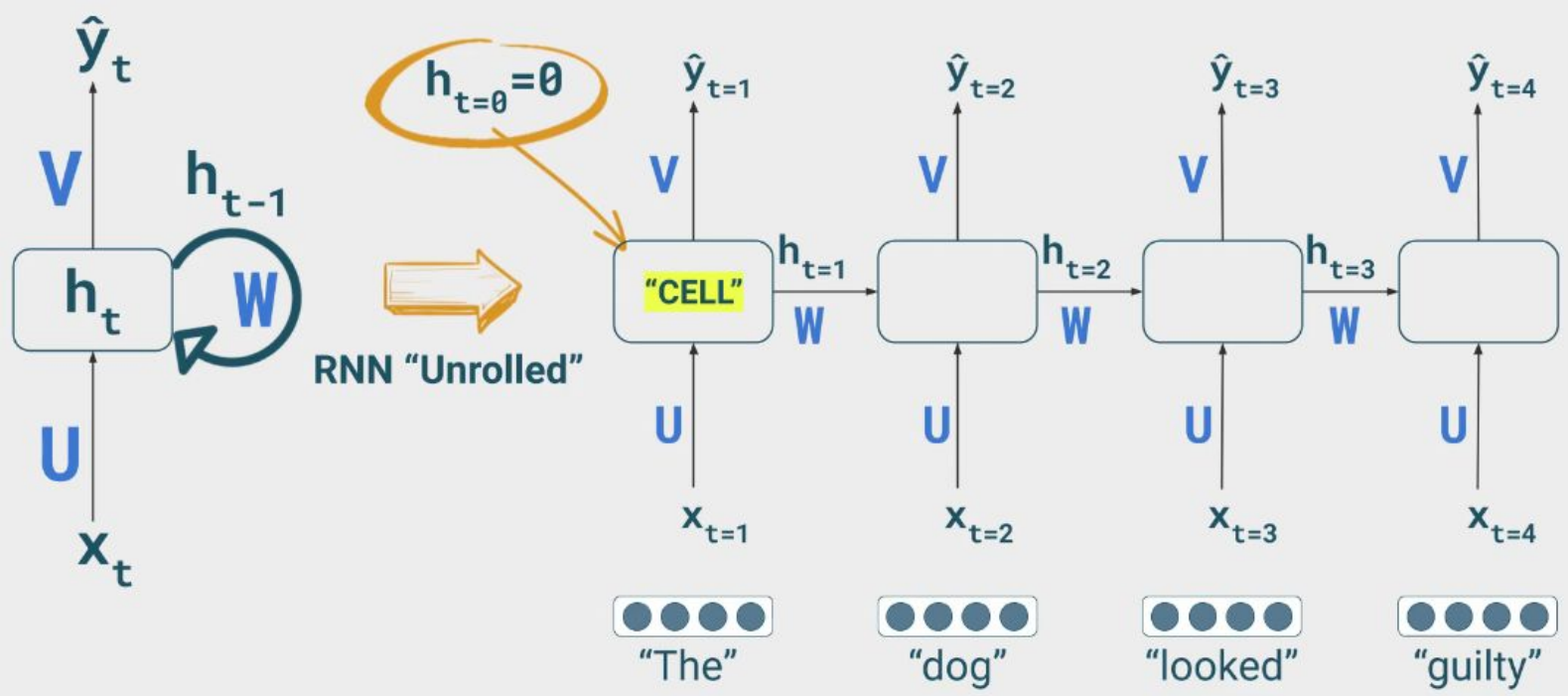


Rekurencyjne sieci neuronowe (RNN)

- **idea:** tekst to sekwencja słów, więc przetwórzmy je po kolei
- **sieci rekurencyjne (RNNs)** przechowują **stan ukryty (hidden state)** i wykorzystują go w przetwarzaniu kolejnych elementów sekwencji
- dla każdego elementu bierzemy poprzedni stan ukryty i aktualne wejście, zwracamy wyjście i przekazujemy stan ukryty dalej

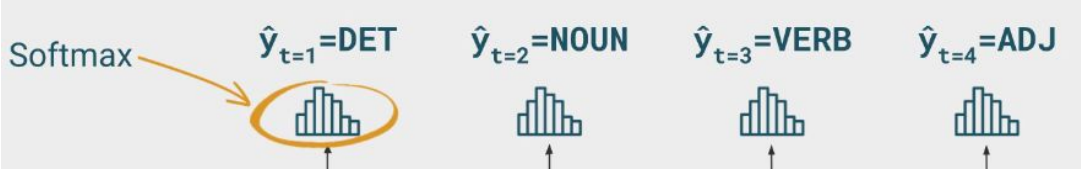


Rekurencyjne sieci neuronowe (RNN)



RNN - cechy

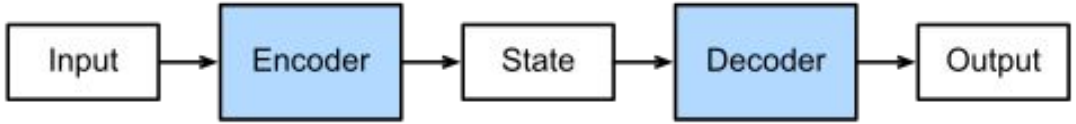
- **wejście:** wektory bezkontekstowe dla tokenów (np. word2vec)
- stan ukryty zawiera informacje o **kontekście** - zdanie aż do aktualnego tokena
- wyjścia to **wektory kontekstowe** - biorą pod uwagę lewy kontekst zdania, np. do klasyfikacji tokenów



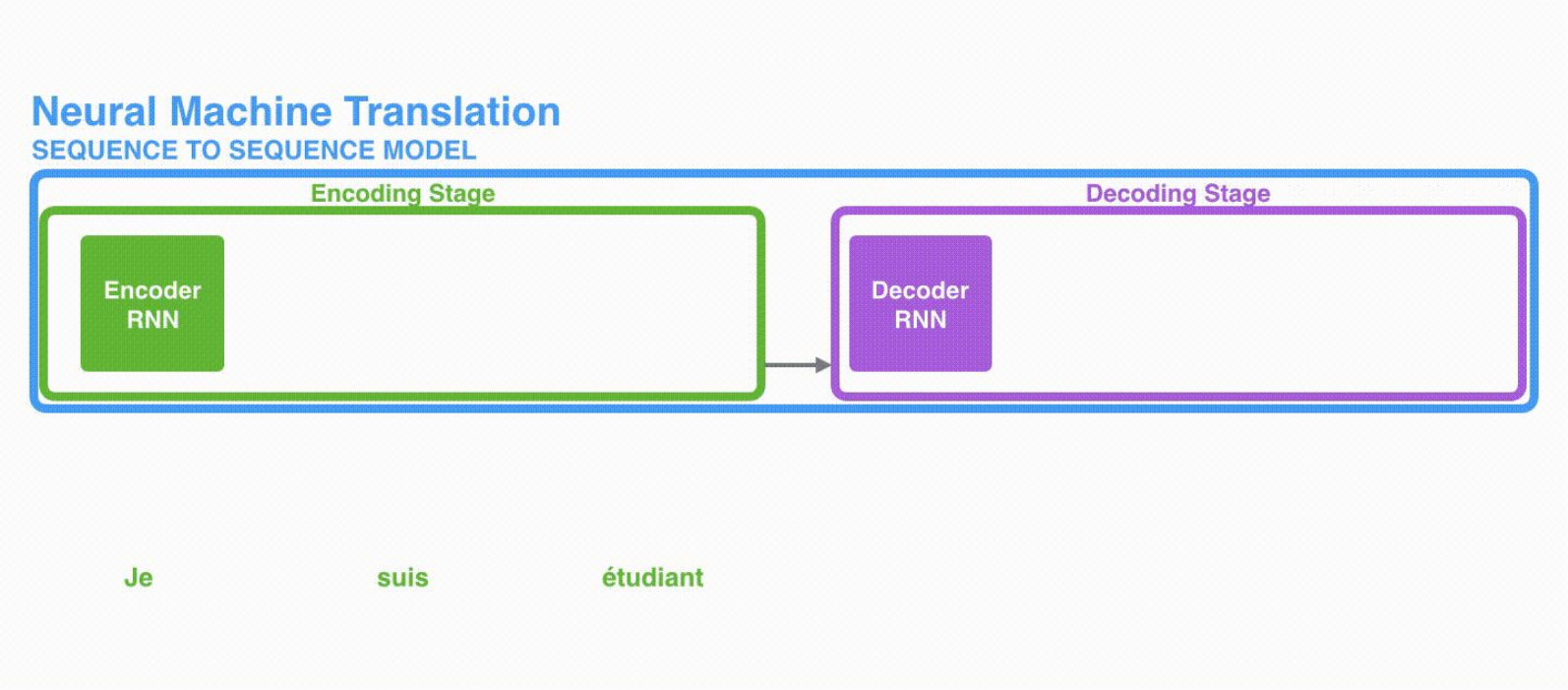
- ostatni hidden state reprezentuje **wektor zdania (sentence embedding)**, np. do klasyfikacji tekstu

Modele seq2seq (enkoder-dekoder)

- tłumaczenie maszynowe to zadanie **sequence to sequence (seq2seq)**
- **idea:** skoro wyjściowy hidden state to reprezentacja zdania w danym języku, to druga sieć da radę "wyciągnąć" z niej te informacje i zwróci drugi język
- pierwsza sieć to **enkoder**, a druga **dekoder**



Modele seq2seq (enkoder-dekoder)



Problem z RNN

- **problemy:**
  - długie łańcuchy zależności - RNN musi zapamiętać całą sekwencję do danego momentu
  - zbyt mała pojemność - trzeba skompresować całe zdanie do 1 wektora
  - tylko lewy kontekst - problem np. przy szyku przestawnym
- **rozwiązania:**
  - lepsze komórki (LSTM, GRU)
  - sieci dwukierunkowe, sieci wielopoziomowe (stacked)
  - **atencja**

Atencja

- **atencja (attention)** - sposób ważenia wejść, aby sieć mogła skupić się na ważnej dla danego tokenu części kontekstu
- seq2seq - enkoder bez zmian, ale przekazuje **wszystkie** stany ukryte do dekodera

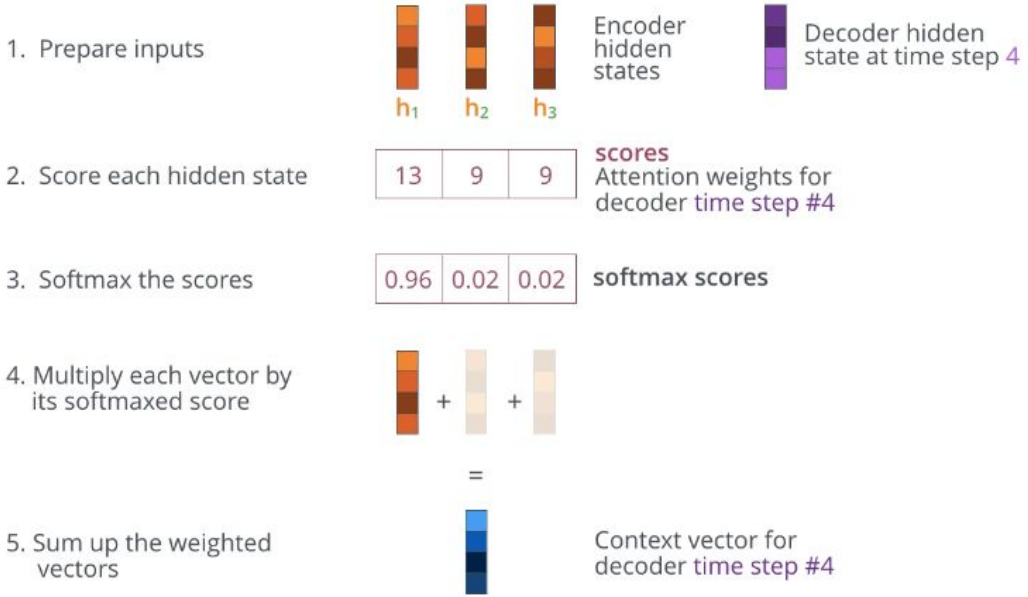
- każdy token dekodera może użyć **ważonych kontekstów** z wielu miejsc

Atencja

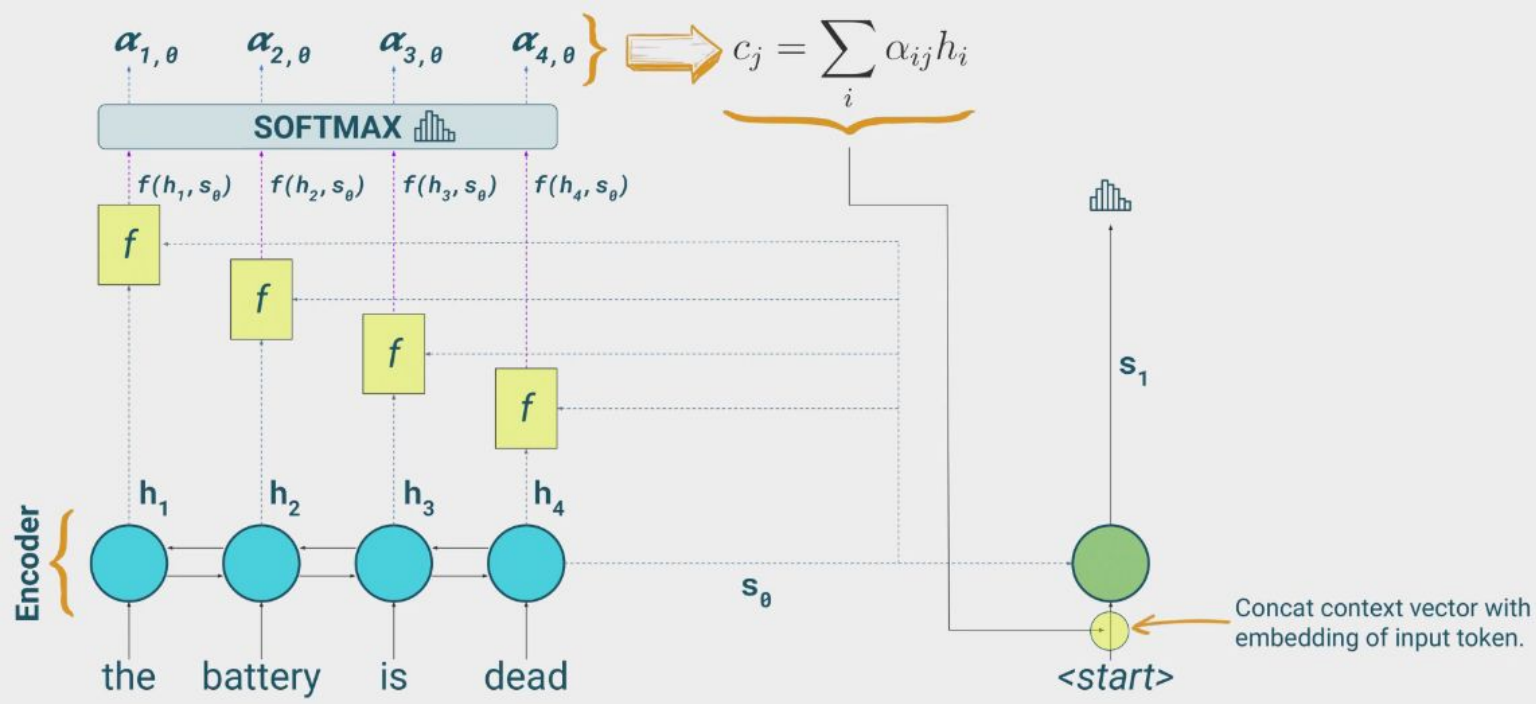
- można zdefiniować bardzo różne wzory, byle miały **wagi** - były **nauczalne (learnable)**

Additive/Bahdanau Attention

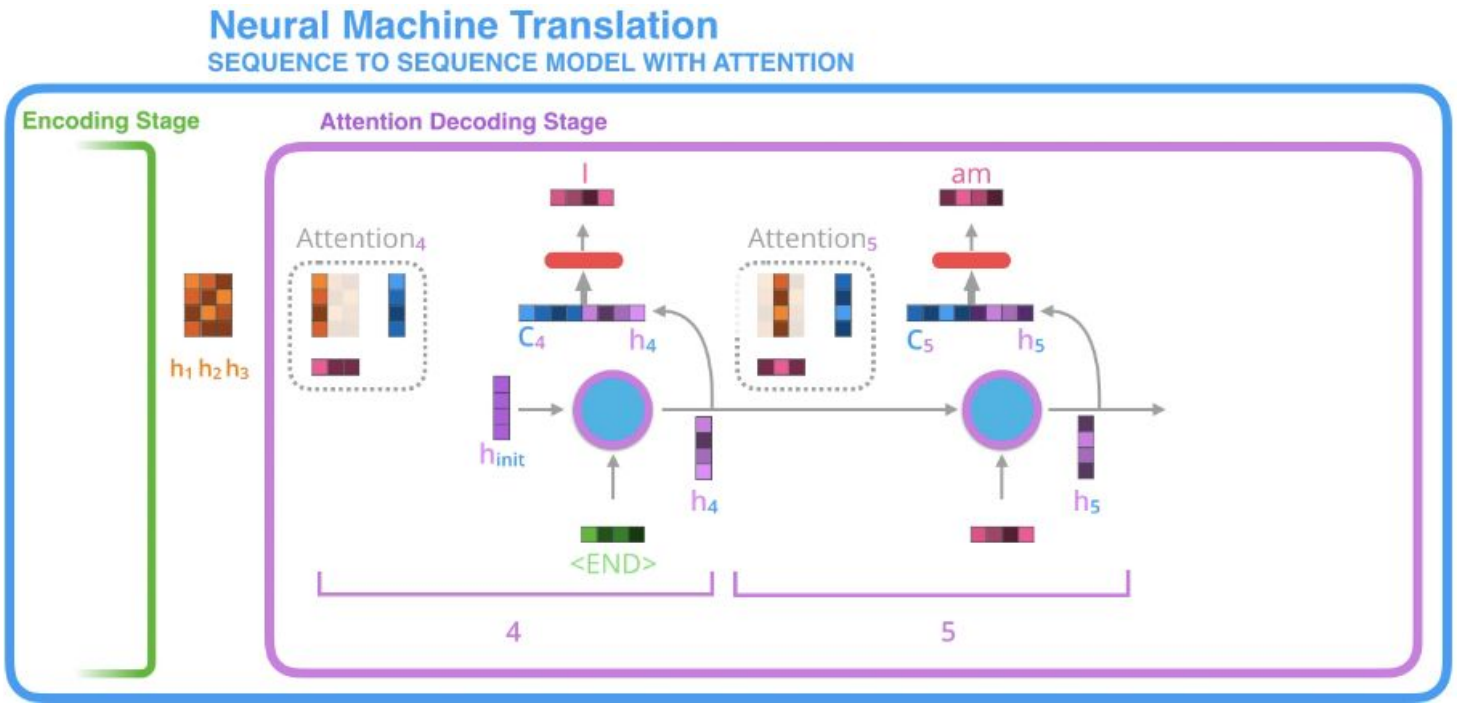
$$f(h_i, s_j) = \mathbf{v}^T \tanh(\mathbf{W}\boldsymbol{h}_i + \mathbf{U}\boldsymbol{s}_j)$$



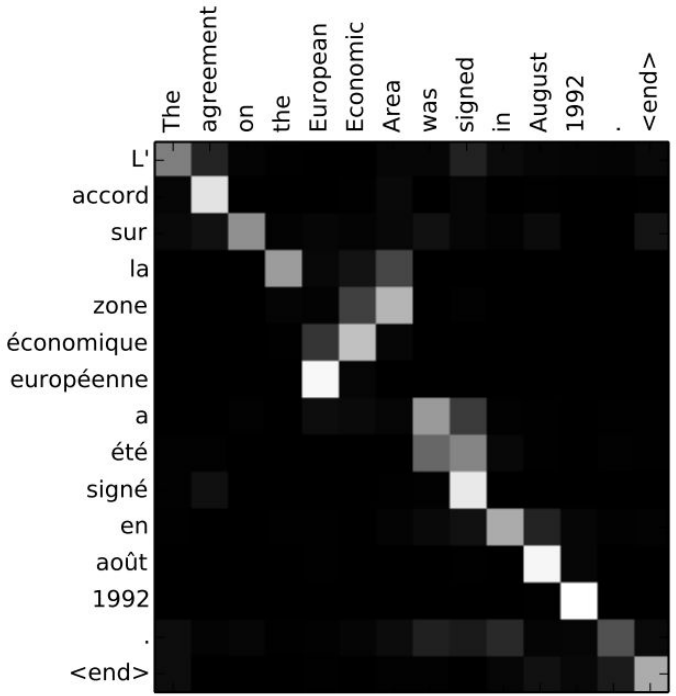
Atencja


$\alpha_{1,\theta}$ $\alpha_{2,\theta}$ $\alpha_{3,\theta}$ $\alpha_{4,\theta}$
$c_j = \sum_i \alpha_{ij} h_i$
SOFTMAX
$f(h_1, s_\theta)$ $f(h_2, s_\theta)$ $f(h_3, s_\theta)$ $f(h_4, s_\theta)$
f
Encoder
h_1 h_2 h_3 h_4
s_θ
s_1
the battery is dead
<start>
Concat context vector with embedding of input token.

Atencja


Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL WITH ATTENTION
Encoding Stage
Attention Decoding Stage
h1 h2 h3
Attention4
Attention5
I
am
C4
h4
C5
h5
hinit
h4
h5
<END>
4
5

Atencja


The
agreement
on
the
European
Economic
Area
was
signed
in
August
1992
.
<end>
L'
accord
sur
la
zone
économique
européenne
a
été
signé
en
août
1992
.
<end>

Dalej problemy

- atencja pomaga, ale nie rozwiązuje wszystkich problemów
- długie łańcuchy zależności dalej nie działają
- RNNy są sekwencyjne - **bardzo wolny trening**
- raczej kiepski transfer learning

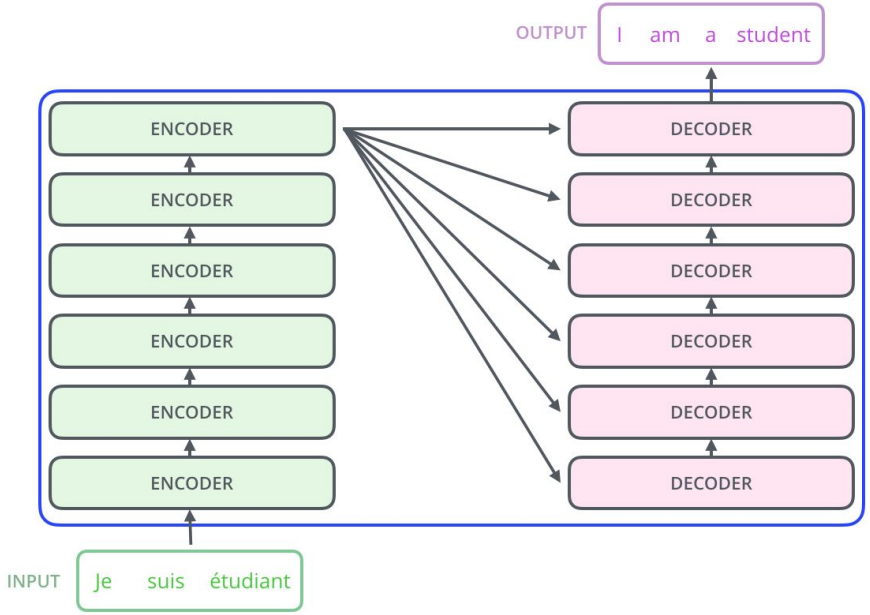
Transformer

"Attention is all you need" Vaswani et al.

- **idea:** skoro atencja jest taka super, a RNN problematyczny, to zostawmy samą atencję
- **każde słowo atenduje z każdym** - każdy ma do dyspozycji cały kontekst
- **zalety:**
  - brak łańcuchów zależności
  - brak sekwencyjności - przetwarzamy całe wejście naraz
- architektura enkoder-dekoder
- **budowa warstwowa** - wyjście ze stacku enkoderów to wejście do każdego enkodera

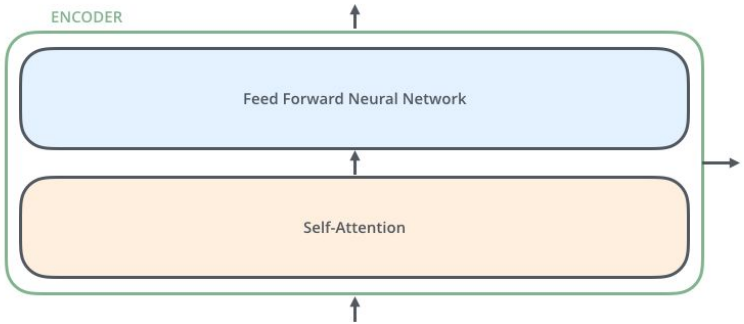
Transformer

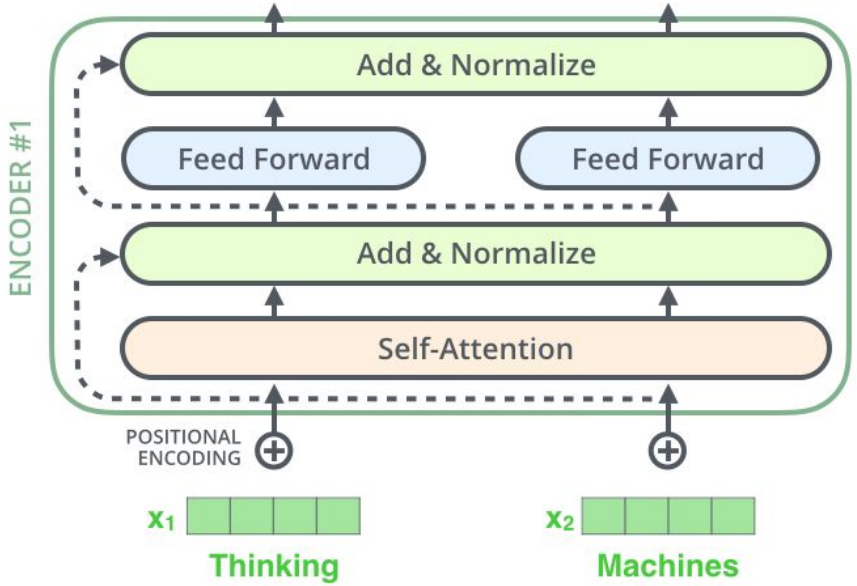
- **seq2seq**
- tłumaczenie maszynowe
- **next token prediction (NTP)** - przewidywanie kolejnego słowa w wyjściowym tekście



Transformer - budowa enkodera

- **self-attention** - każde słowo atenduje z każdym z własnego zdania
- **feedforward NN** - zwykły MLP, żeby była nieliniowość

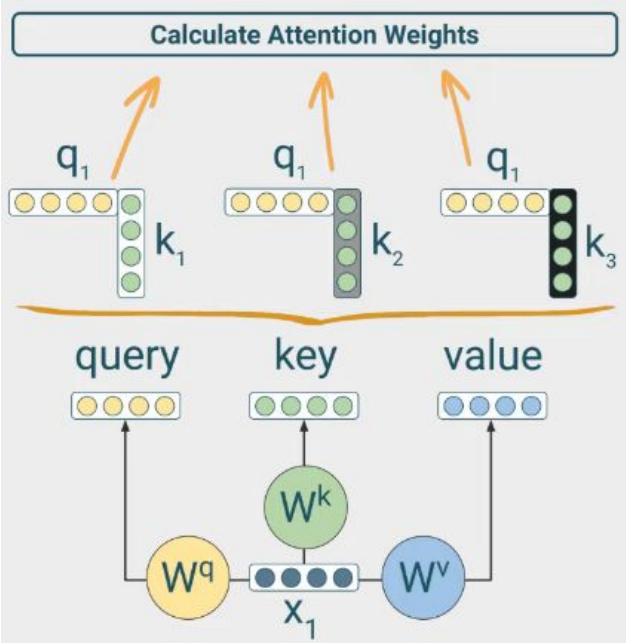




Key-value attention

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$

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



QK^T - iloczyn skalarny mówi, jak bardzo danemu tokenowi (query) są przydatne inne tokeny (key) - włącznie z nim samym!

sqrt(d_k) - żeby nie nasycić gradientów dla wielu wymiarów

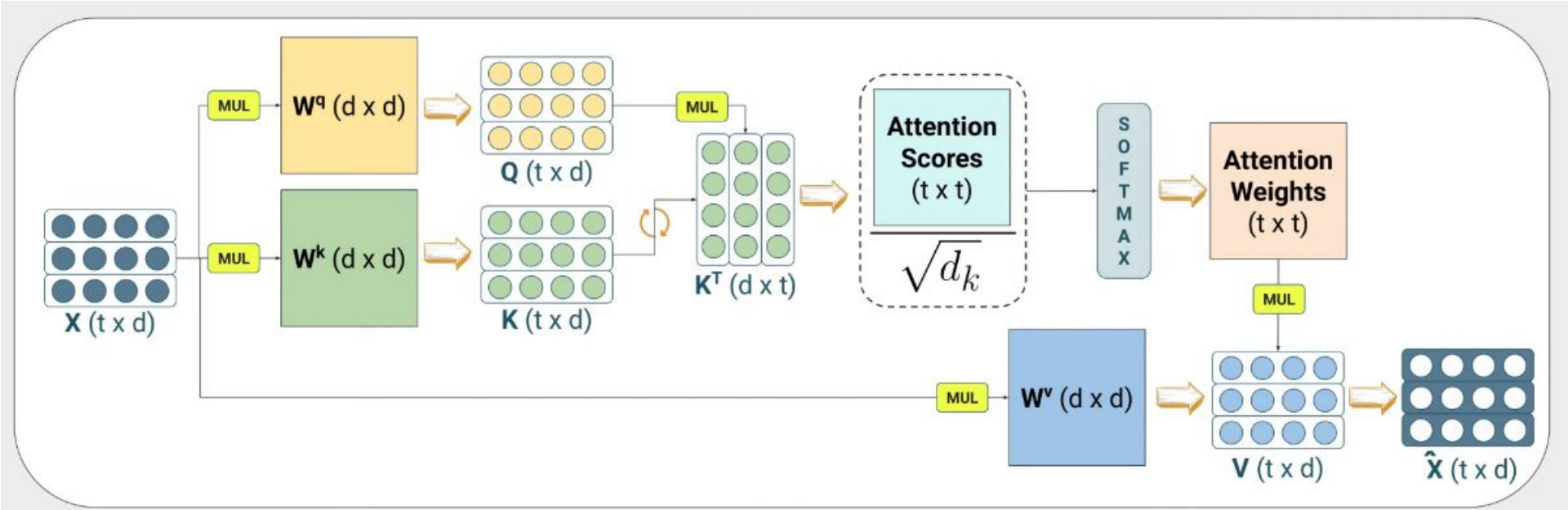
softmax - żeby sumowało się do 1

V - mnożymy dany token (value) przez wagi atencji

Key-value attention

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$

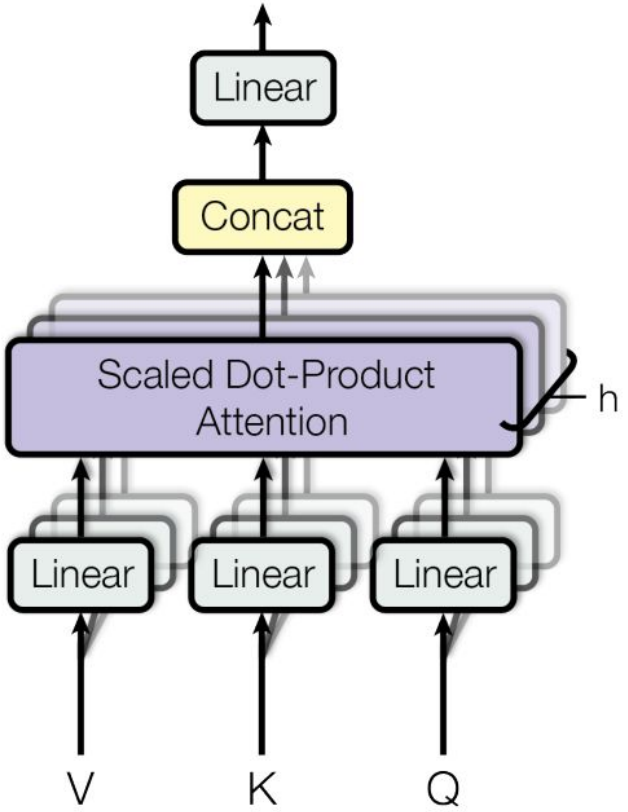
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



Multihead attention

- **obserwacja:** słowo może zależeć od wielu innych słów
- **idea:** zróbmy wiele atencji równolegle, żeby można było atendować do różnych rzeczy
- pojedyncza atencja to **attention head**, robimy 8 **równolegle**
- każdy head jest ma 8 razy mniejszą wymiarowość (512 / 8 = 64), żeby koszt obliczeniowy został prawie identyczny
- **ważone łączenie** - po prostu warstwa liniowa

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_\mathrm{h})W^O$$

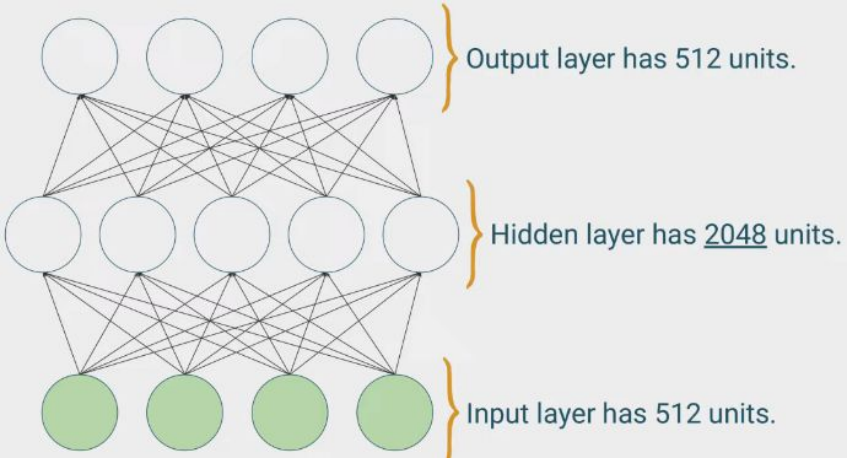


Feedforward NN

- **obserwacja:** atencja jest wszędzie liniowa
- **idea:** zróbmy nieliniowość z użyciem małego MLP
- liniowa + ReLU + liniowa

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

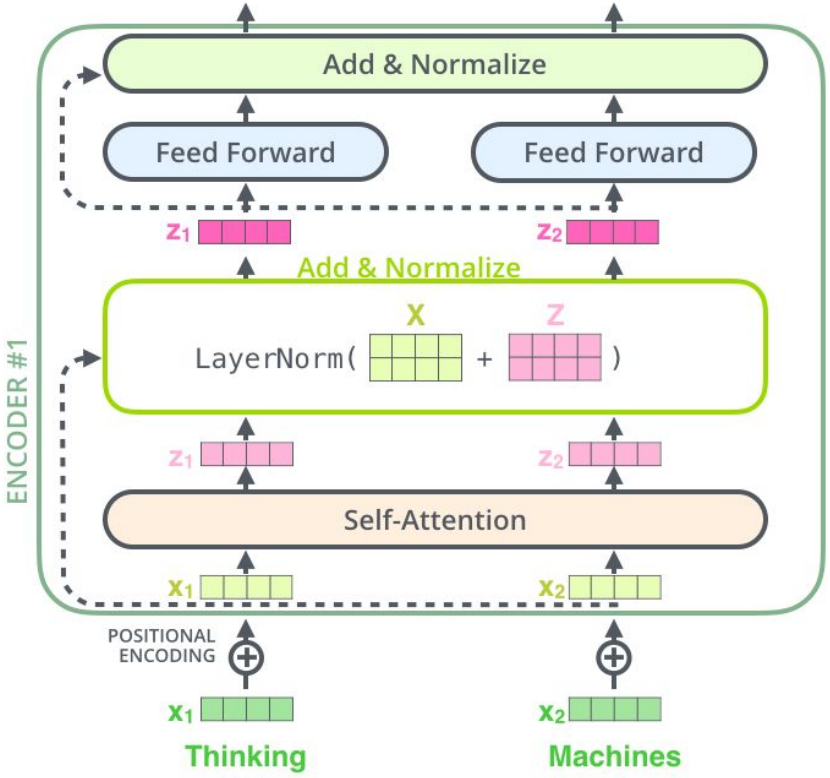
- sieć **szeroka** w środku, żeby mieć "mocną" nieliniowość
- rzut liniowy, żeby nie zmieniać wymiaru na wyjściu



Blok enkodera

- wiemy już, jak działa self-attention i MLP
- dodajemy **połączenia rezydualne** dookoła atencji i MLP - idea jak w sieci ResNet
- dodajemy **layer normalization (LayerNorm)** dla stabilizacji:
  - standaryzacja per tekst
  - skalowanie i przesunięcie - learnable

$$\mathrm{LayerNorm}(x) = \alpha \left(\frac{x - \mu}{\sigma} \right) + \beta$$

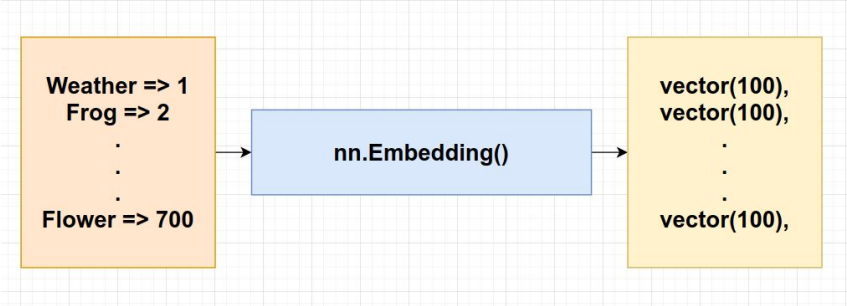


Enkoder

- seria bloków jeden po drugim: self-attention + normalizacja + MLP + normalizacja
- **obserwacja:** po usunięciu RNN straciliśmy informację o kolejności słów!
- atencja - każdy z każdym, MLP - naturalnie bez kolejności
- **trzeba dodać informację o kolejności**

Wejście do transformera

- **wejście w RNN:** pretrenowane wektory słów
- embeddingi słów w transformerze są **uczone**
- mamy dany zbiór tokenów (vocabulary), np. 30000
- one-hot encoding + rzut liniowy na wymiar 512
- `nn.Embedding` - "słownik" w PyTorchu, wrzucamy indeks, w środku robi one-hot encoding i rzut liniowy



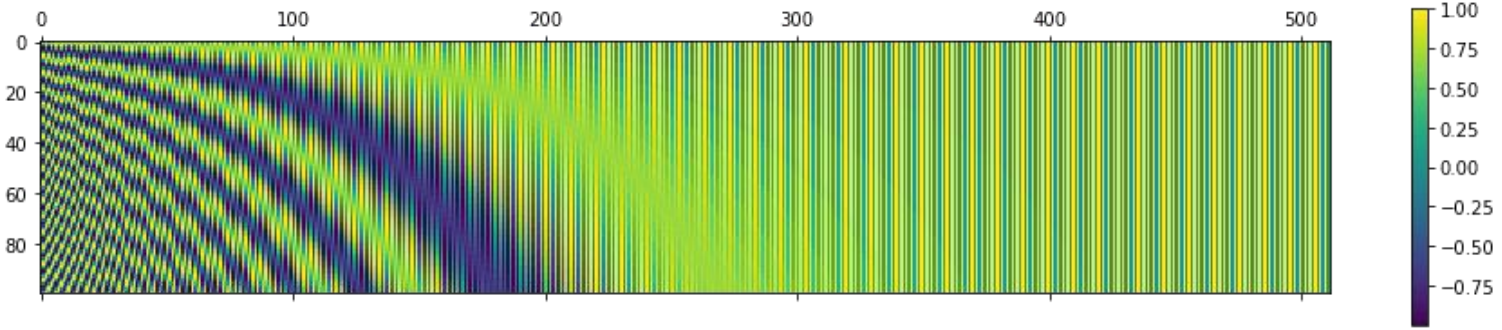
Positional encodings

- **idea:** weźmy funkcję matematyczną, która będzie inna na każdej pozycji

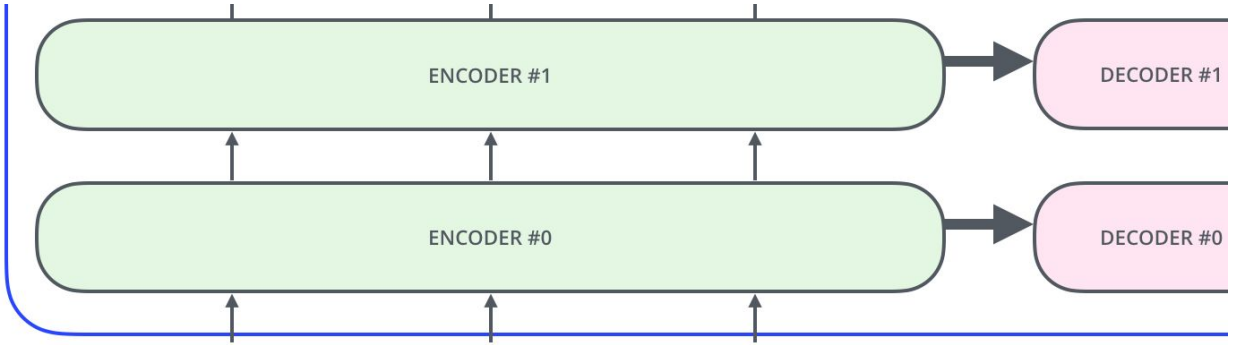
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

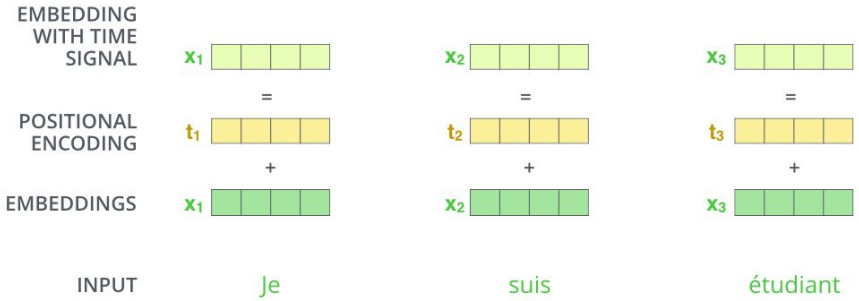
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

- **sumujemy** embedding z samych słów i z pozycji



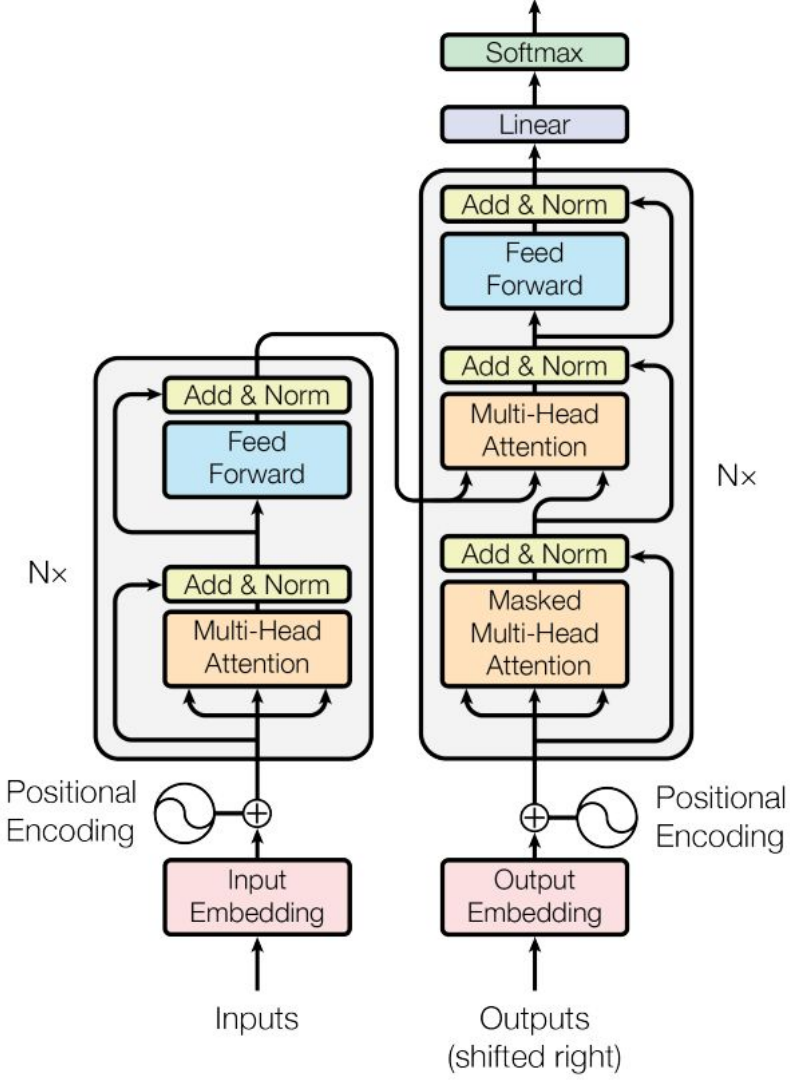
Positional encodings





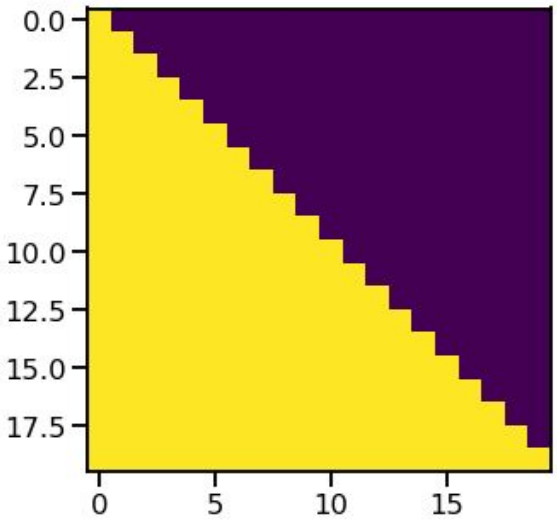
Dekoder

- **wejście podczas treningu:** poprzednie docelowe tokeny (przesunięte o 1 w prawo)
- **wejście podczas predykcji:** poprzedni token
- **maskowana atencja** - nie możemy patrzeć na przyszłe tokeny podczas tłumaczenia, więc robimy atencję -∞ wszędzie na prawo (tylko lewy kontekst)
- **atencja enkoder-dekoder** - wejściem do query Q i kluczy K są wyjścia z enkodera (wartości normalnie z dekodera)
- **wyjście:** wektor dla kolejnego tokena w tekście, robimy softmax i bierzemy słowo z najwyższym prawdopodobieństwem

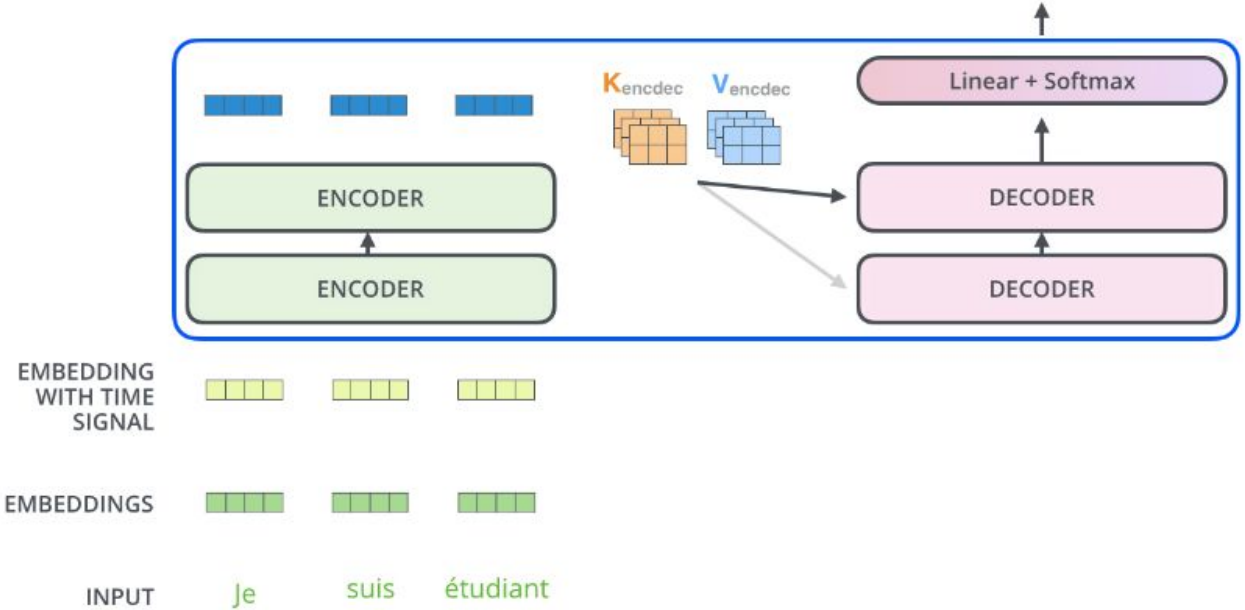


Dekoder

Maska atencji



Atencja enkoder-dekoder

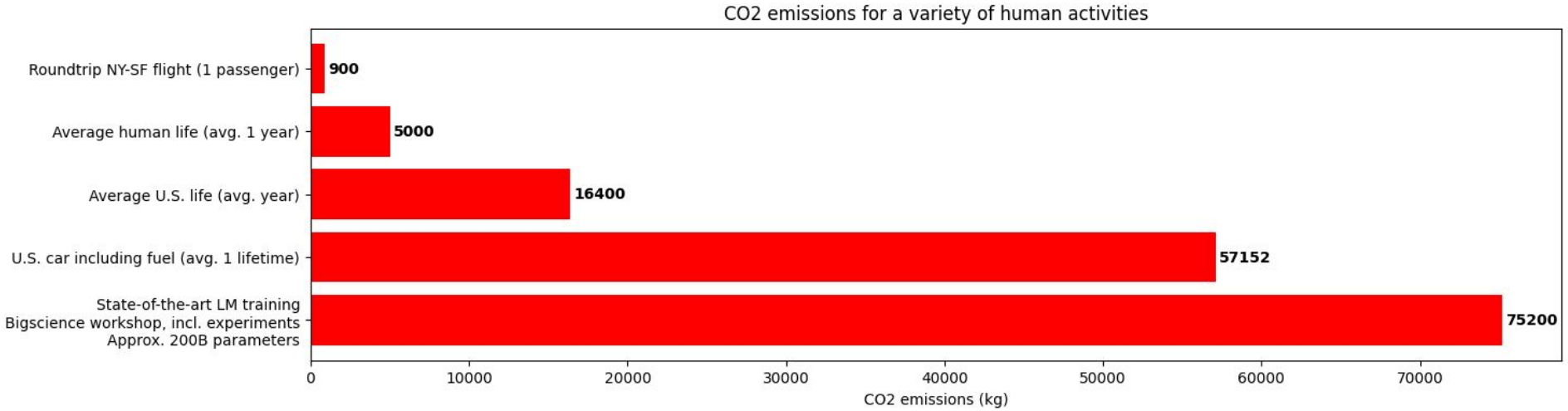


Inne szczegóły

- **byte-pair encoding (BPE)** - tokenizacja na podsłowa (subword tokenization), vocabulary ok. 32000 / 37000 tokenów
- **dropout** - delikatny (0.1), po embeddingach i po każdej sub-warstwie (atencja, MLP)
- **label smoothing** - "rozmyte" docelowe klasy zamiast jednoznacznego ground truth

Koszt obliczeniowy

Czas treningu: 12h (base), 3.5 dnia (large)



Architektury transformerowe

Encoder-decoder

- **zastosowanie:** zadania sequence-to-sequence
- np. machine translation, text summarization
- **modele:** oryginalny transformer, BART, T5, ProphetNet, …

Encoder-only

- **zastosowanie:** zadania dyskryminatywne
- np. klasyfikacja tekstów / tokenów, extractive QA
- specyficzny pretrening, zasadniczo zawsze wymaga fine-tuningu na docelowym zadaniu
- **modele:** BERT, RoBERTa, HerBERT, DistilBERT, ALBERT, ELECTRA, SentenceBERT, …

Decoder-only

- **zastosowanie:** zadania generatywne
- np. generowanie tekstu
- **modele:** GPT, GPT-2, GPT-3, TransformerXL, CTRL

BERT

BERT

- **Bidirectional Encoder Representation from Transformers**
- model encoder-only
- **obserwacja:** jest wiele zadań, gdzie zawsze mamy cały kontekst (np. klasyfikacja tekstów), więc można wyrzucić dekoder
- **chcemy:** taniego treningu i dobrych zdolności transfer learningu
- **idea:** nauczmy model nie specyficznego zadania (tłumaczenie maszynowe), tylko ogólnej **reprezentacji języka**, żeby się dobrze generalizował

BERT unsupervised pretraining

- **idea:** zróbmy taki pretrening, żeby był bez nadzoru, bo wtedy możemy wykorzystać dowolne zbiory tekstów
- BERT jest trenowany na 2 zadaniach jednocześnie:
  - **masked language modelling (MLM)** - losowo maskujemy słowo i model ma przewidzieć, co tam powinno być
  - **next sentence prediction (NSP)** - uczymy zawsze na 2 sklejonych ze sobą zdaniach i model ma przewidzieć, czy następowały one oryginalnie po kolei

Masked language modelling (MLM)

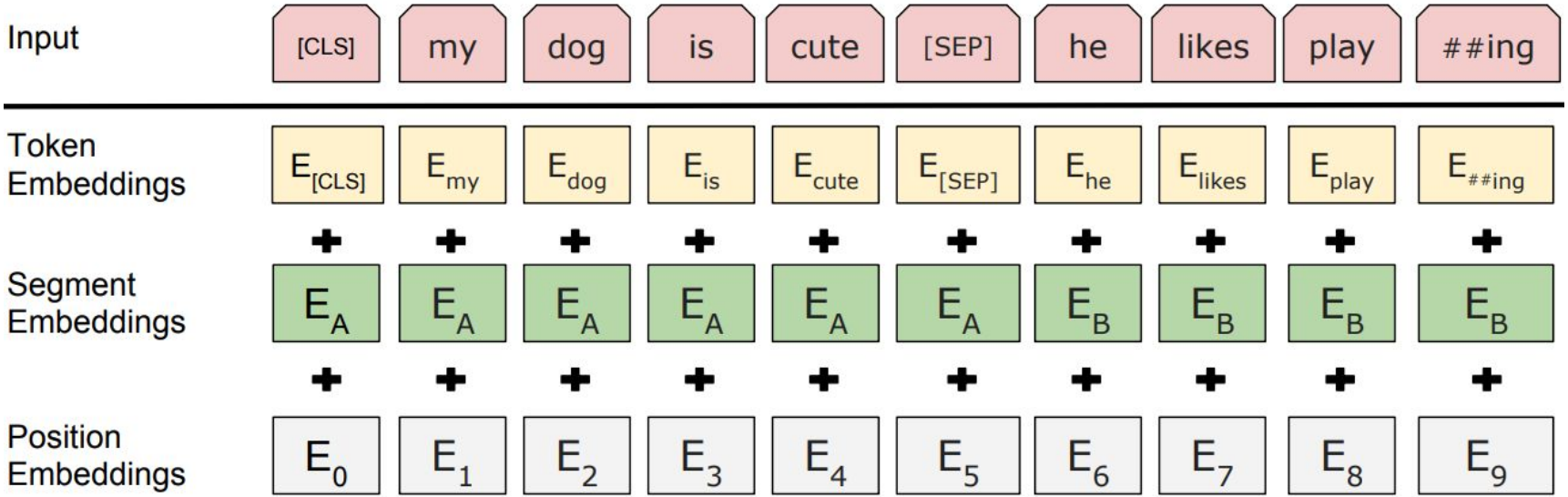
- **idea:** niech model nie przewiduje kolejnego tokenu, tylko losowo wybrany wewnątrz tekstu
- **losujemy 15% tokenów**, zamieniamy je losowo, a model ma je przewidzieć

Czym zastępujemy	Ilość (spośród 15%)	Cel
[MASK]	80%	Nauka słowa na podstawie kontekstu
Losowe słowo	10%	Regularyzacja, trzeba solidnie polegać na kontekście
To samo słowo (brak zmiany)	10%	Nauka, jak dane słowo wygląda w tym kontekście

Next sentence prediction (NSP)

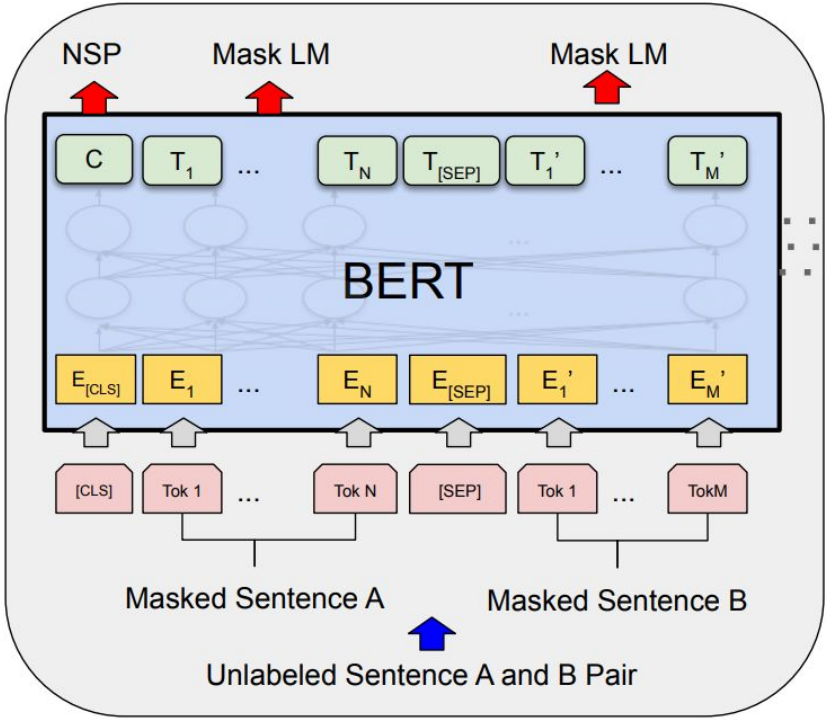
- **obserwacja:** wiele zadań wymaga zrozumienia relacji 2 zdań, np. question answering
- **idea:** niech model przewiduje, czy jedno zdanie jest po drugim w tekście
- zmuszamy model, żeby nauczył się, jak wygląda spójna wypowiedź, w jaki sposób zdania wynikają z siebie
- wykorzystuje specjalne tokeny [CLS], [SEP] i dodatkowy wektor wskazujący na zdanie (segment embedding)

BERT tokens


Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
Token Embeddings
E[CLS]
Emy
Edog
Eis
Ecute
E[SEP]
Ehe
Elikes
Eplay
E##ing
Segment Embeddings
EA
EB
Position Embeddings
E0
E1
E2
E3
E4
E5
E6
E7
E8
E9

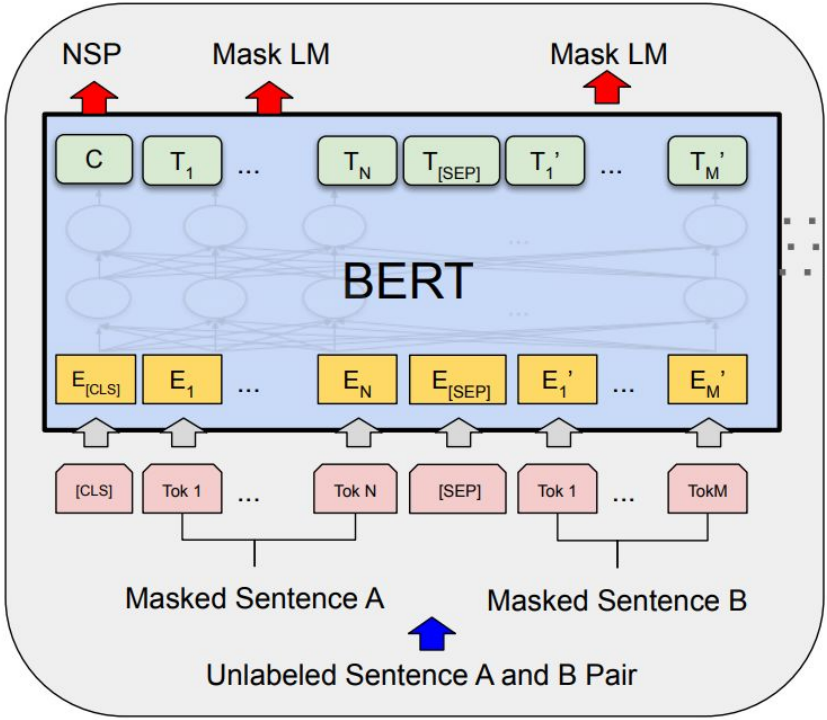
BERT pretraining dataset

- **kopiujemy** zbiór danych 10 razy, za każdym razem losując maskowanie dla każdego zdania
- bierzemy **pary zdań** i sklejamy je ze sobą tokenem [SEP]
- 50% par jest losowych (klasa 0), 50% to kolejne zdania (klasa 1)
- na początek każdej pary **doklejamy** token [CLS]

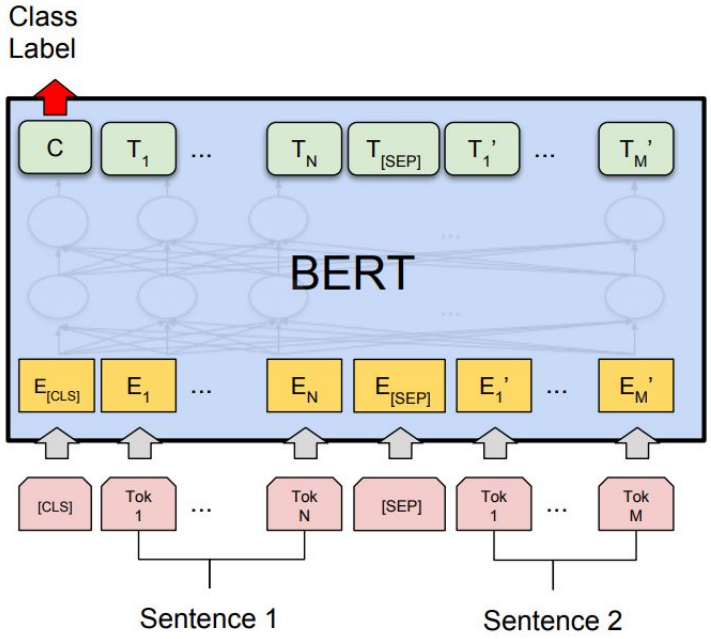


BERT pretraining procedure

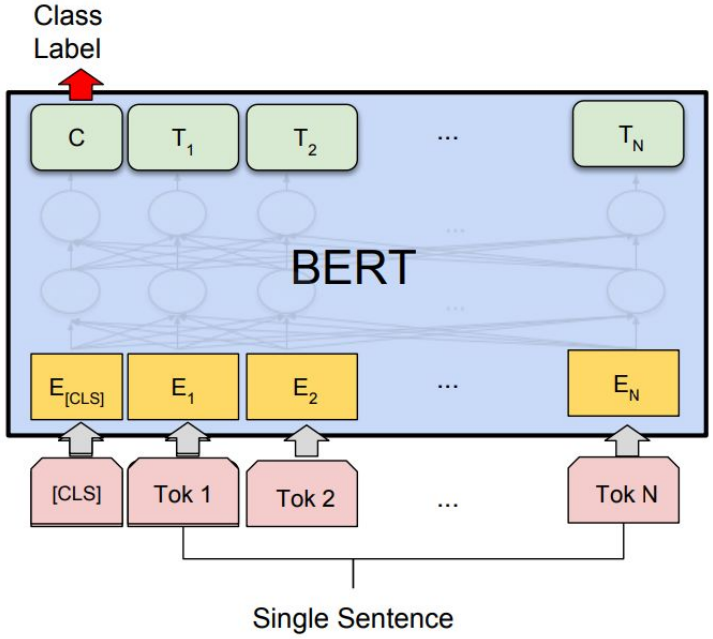
- wrzucamy kolejne zdania do modelu
- wektor [CLS] wrzucamy do MLP, które daje binarną predykcję NSP (kolejne / nie po kolei zdania)
- wektory dla tokenów [MASK] wrzucamy do drugiego MLP, daje predykcję dla każdego tokenu
- [SEP] jest ignorowany
- **funkcja kosztu:** suma kosztów MLM i NSP (oba to entropia krzyżowa)



BERT finetuning

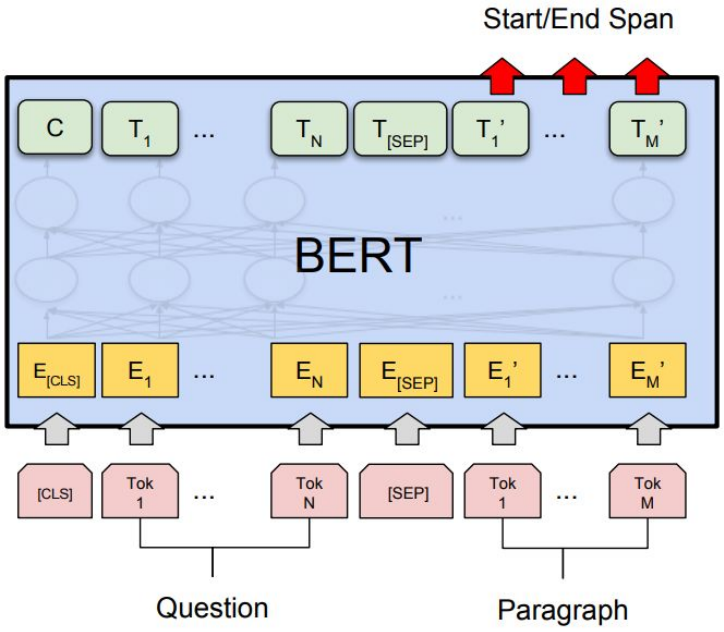


(a) Sentence Pair Classification Tasks:

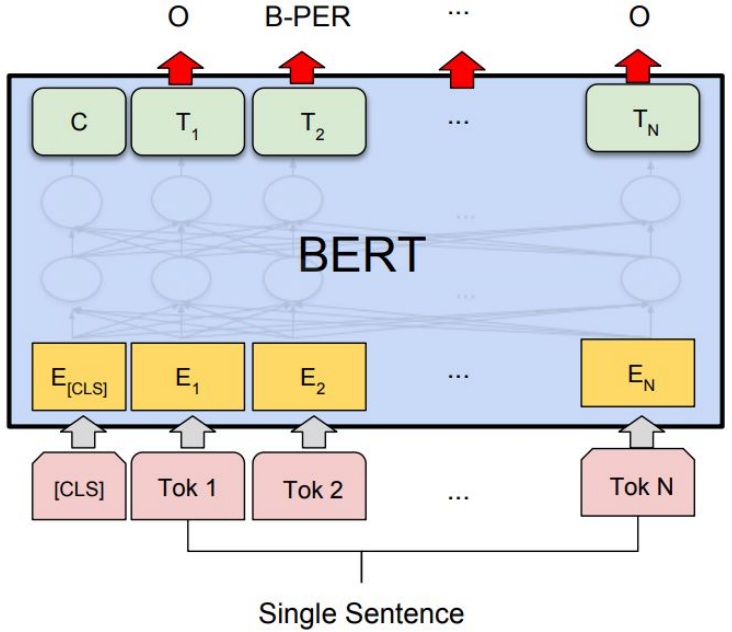


(b) Single Sentence Classification Tasks:

BERT finetuning



(c) Question Answering Tasks:



(d) Single Sentence Tagging Tasks:

Tokenizacja w BERTcie

- algorytm **WordPiece** - proprietary Google'a, ale są implementacje open source
- podobny do Byte Pair Encoding (BPE), ale z pewnymi drobnymi różnicami
- generalnie takie same wyniki

Ważne pochodne BERTa

RoBERTa:

- bardzo podobny model
- taki sam rozmiar co BERT
- liczne usprawnienia treningu, szczególnie o wiele więcej danych
- brak NSP, samo MLM
- zazwyczaj lepsze wyniki

ALBERT:

- dużo mniejszy od BERTa (base - ok. 9x)
- współdzielenie wag pomiędzy warstwami
- Sentence Order Prediction (SOP) zamiast NSP - negatywne przykłady to 2 zdania w złej kolejności
- zazwyczaj lepsze wyniki od BERTa i RoBERTy

Ważne pochodne BERTa

HerBERT:

- BERT, ale dla języka polskiego
- transfer learning, zaczyna od wag z RoBERTy
- zmodyfikowane MLM + Sentence Order Prediction (dość podobne do NSP, klasy: następne / poprzednie / niezwiązane zdanie)

DistilBERT:

- 2 razy mniejszy od BERTa
- trenowany w reżimie **knowledge distillation** - mały model (student) uczy się odwzorować duży model (nauczyciela)
- daje wyniki bardzo zbliżone do BERTa, a jest sporo szybszy

Ważne pochodne BERTa

SentenceBERT:

- zwykły BERT rozumie dobrze semantykę całych zdań tylko do zdań dyskryminatywnych, po fine-tuningu
- do zadań nienadzorowanych (np. semantic search, text clustering) jest bardzo kiepski, często o wiele gorszy od użycia GloVe, a do tego niesamowicie wolny
- trenowany w reżimie **sieci syjamskich (siamese networks)** - dotrenowujemy BERTa, ucząc go dodatkowo dużej odległości między różnymi przykładami, a małej między podobnymi

Pytania?