Dekodowanie. Trenowanie modeli

Paweł Rychlikowski

Instytut Informatyki UWr

30 października 2024

Ocena modeli językowych

Są generalnie dwa sposoby oceniania modeli językowych (i, tak naprawdę, wszystkiego innego też):

1. Wewnętrzna (instristic) – mamy jakąś mniej lub bardziej naturalną miarę jakości modelu
2. Zewnętrzna (exstrinsic) – sprawdzamy, jak model poradzi sobie z pewnym zadaniem (które jest naszym celem, i w którym mamy naturalną miarę jakości)

Z miarą zewnętrzną spotykamy się od pierwszej pracowni.

Perplexity (miara nieokreśloności)

Intuicje

1. To co się zdaża, powinno mieć wysokie prawdopodobieństwo.
2. Gdy dobrze przewidujemy kolejne słowo (na podstawie pełnego prefiksu), to jesteśmy w stanie dobrze kompresować tekst (dlaczego?)
3. Oczywiście powinniśmy dzielić korpus (na część przeszłą (zdarzyła się) i przyszłą (zdarzy się, chcemy jej dać spore prawdopodobieństwo, ale jej nie znamy)

Perplexity (2)

Wzór

$$PP(w_1 \dots w_N) = P(w_1 \dots w_N)^{-\frac{1}{N}}$$

gdzie N jest wielkością części testowej korpusu

Pytanie

Jakie jest perplexity całkiem losowego ciągu cyfr?(odpowiedź: **10**)

Można rozumieć perplexity jako średni ważony **współczynnik rozgałęzienia** języka.

Pytania

- Czy zachłanne dekodowanie maksymalizuje perplexity?
- Czy sampling daje ciągi o dużym perplexity?

Jak lepiej maksymalizować perplexity?

Uwaga

Maksymalizacja perplexity generowanego tekstu nie musi być najlepszym rozwiązaniem. W szczególności powtarzające się teksty mają wysokie perplexity.

Dygresja

- Na tablicy o tłumaczeniu maszynowym, podobieństwa i różnice w stosunku do zwykłego modelowania języka.

Problems with greedy decoding

- Greedy decoding has no way to undo decisions!
  - Input: *il a m'entarté* *(he hit me with a pie)*
  - → *he* ____
  - → *he hit* ____
  - → *he hit a* ____ *(whoops! no going back now…)*
- How to fix this?

29

Exhaustive search decoding

- Ideally, we want to find a (length *T*) translation *y* that maximizes

$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots,P(y_T|y_1,\ldots,y_{T-1},x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1,\ldots,y_{t-1},x)$$

- We could try computing all possible sequences *y*
  - This means that on each step *t* of the decoder, we're tracking V^t possible partial translations, where *V* is vocab size
  - This $O(V^T)$ complexity is far too expensive!

30

Beam search decoding

- Core idea: On each step of decoder, keep track of the *k* most probable partial translations (which we call *hypotheses*)
  - *k* is the beam size (in practice around 5 to 10, in NMT)

- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top *k* on each step

- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!

31

Beam search decoding: example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

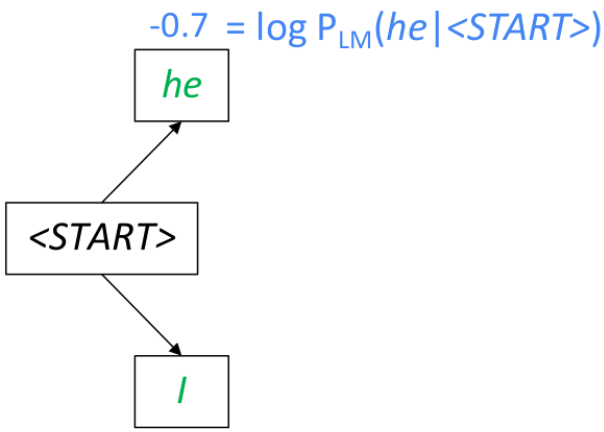
<START>

Calculate prob dist of next word

32

Beam search decoding: example

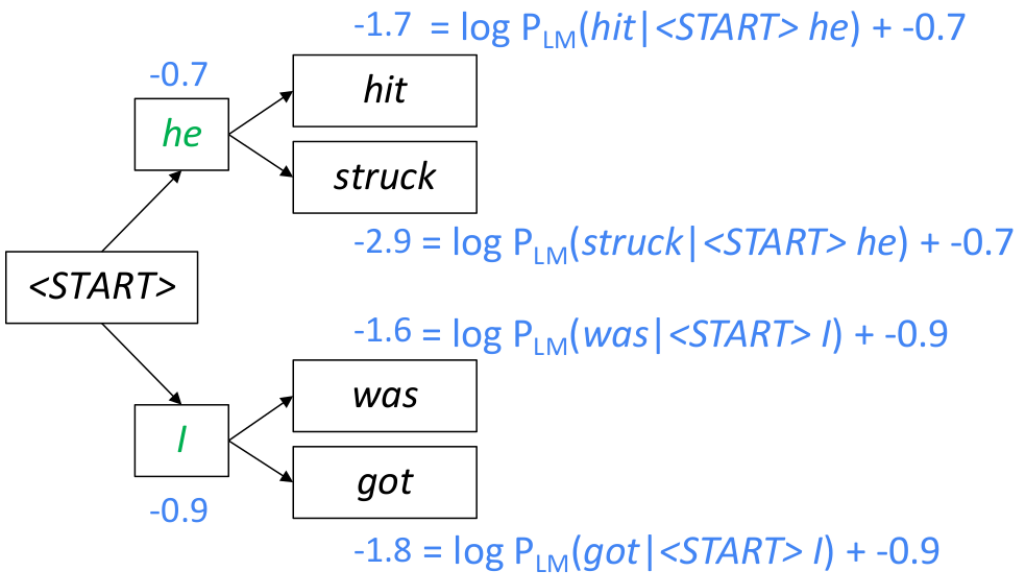
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Take top *k* words and compute scores

Beam search decoding: example

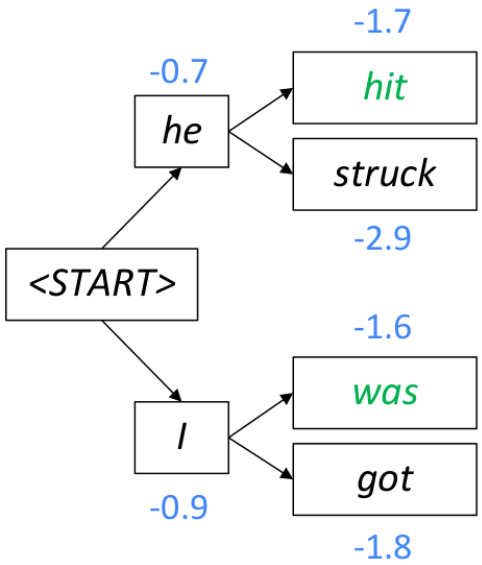
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

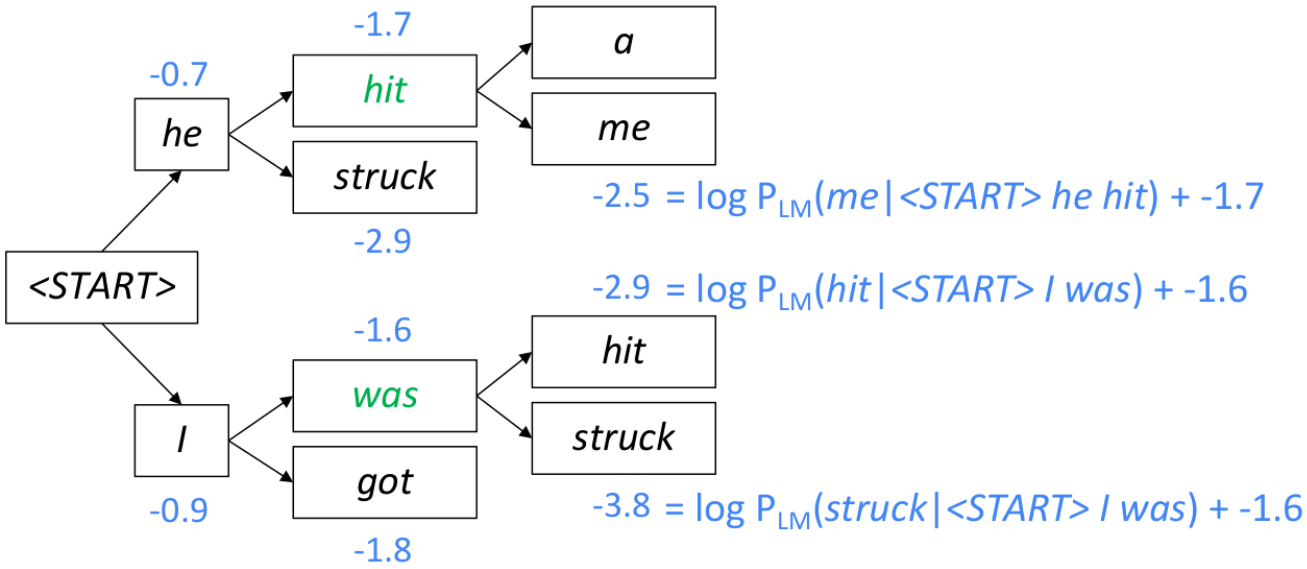
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding: example

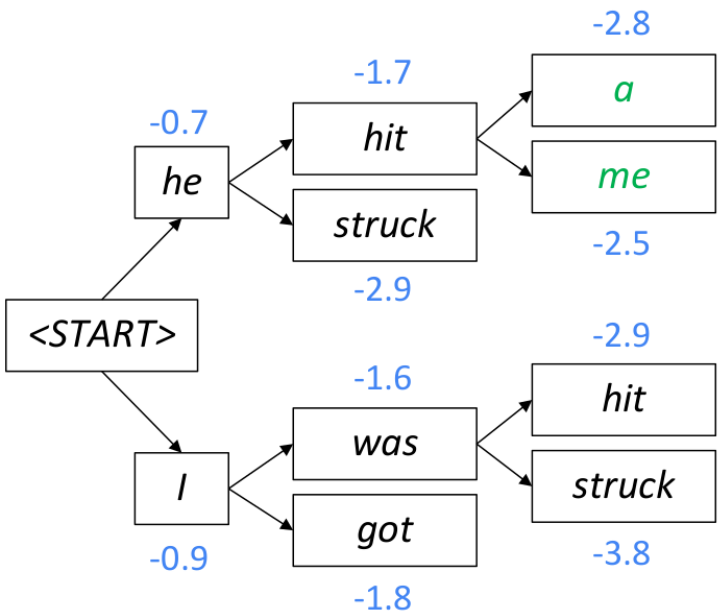
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

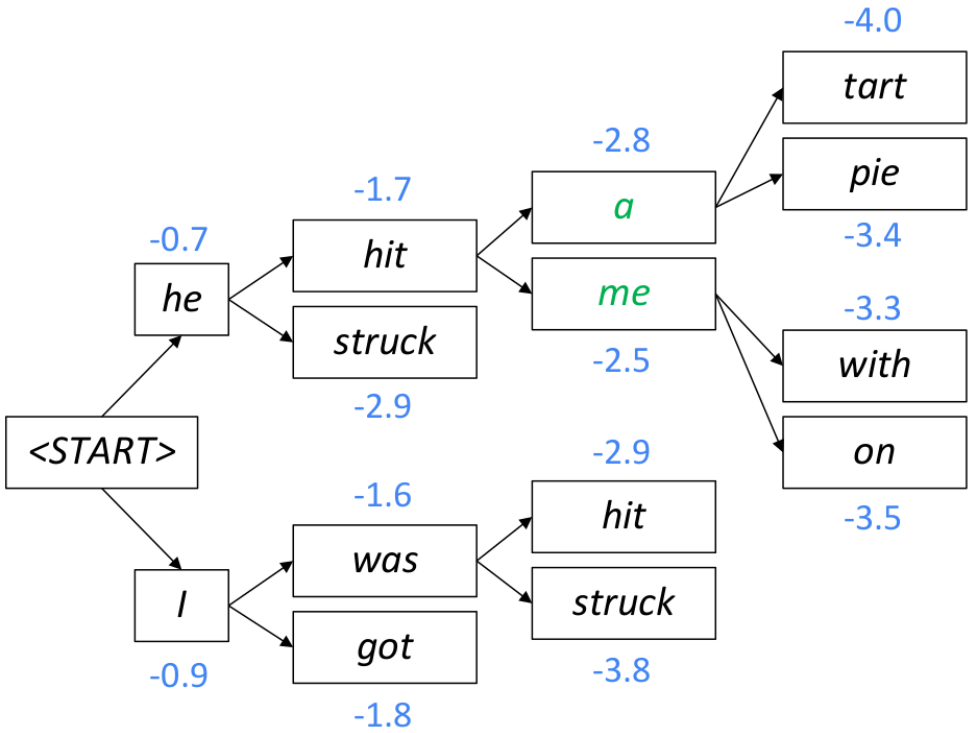
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses, just keep k with highest scores

Beam search decoding: example

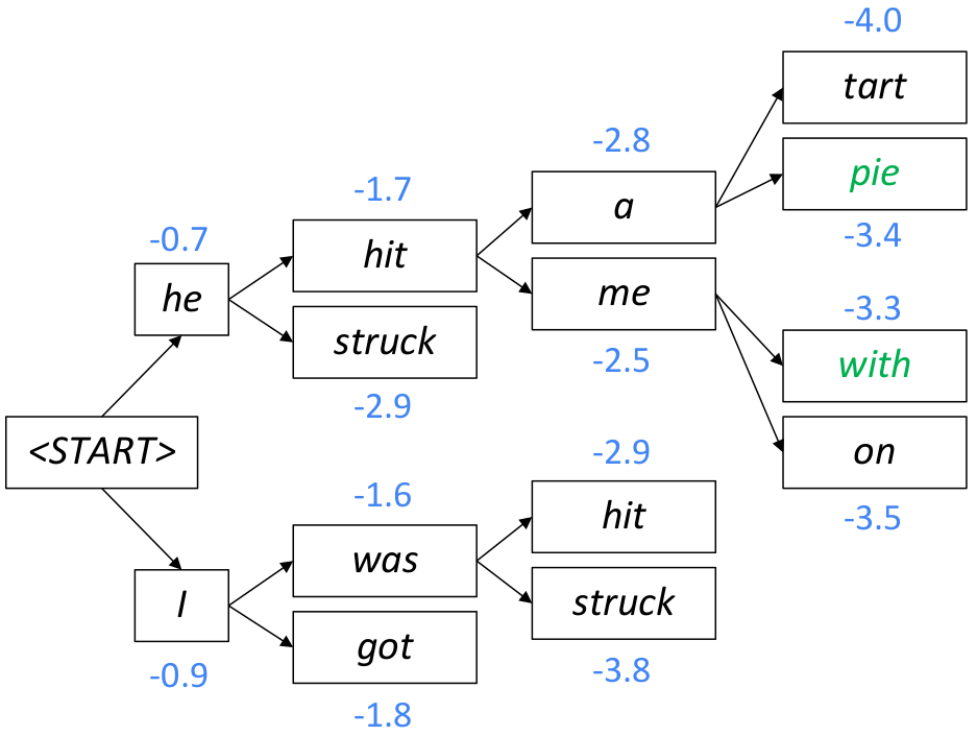
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

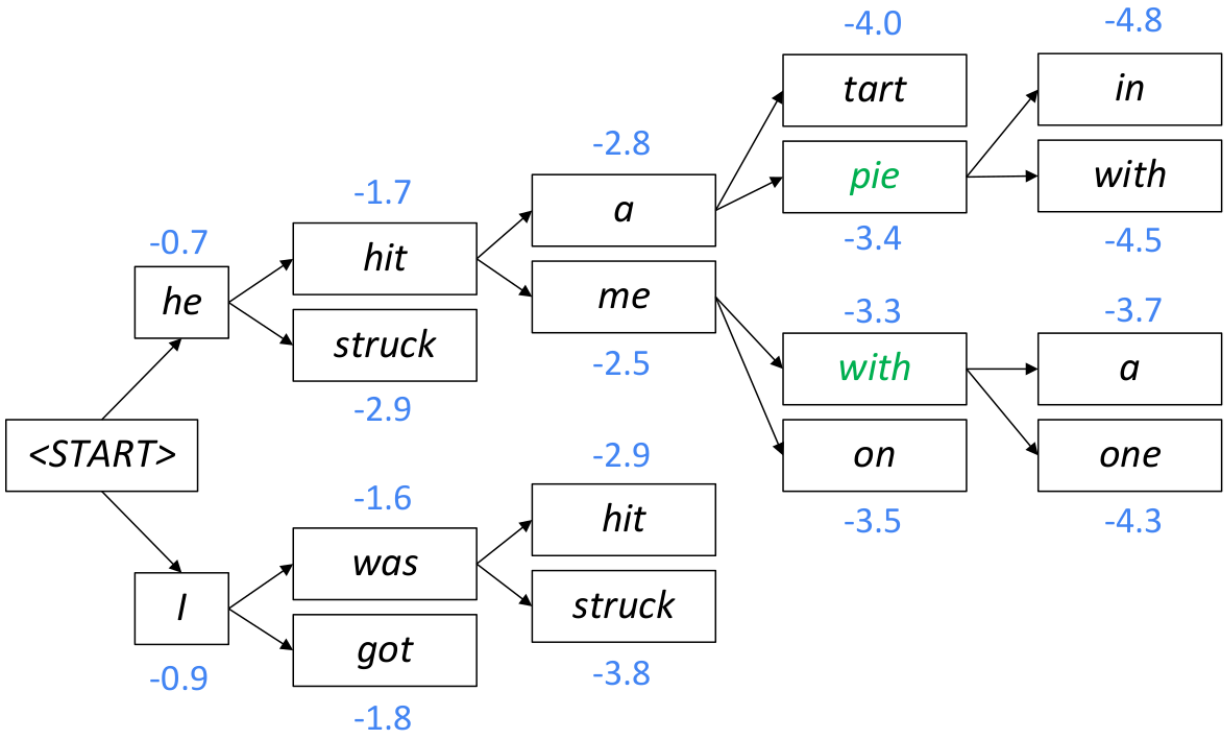
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses, just keep k with highest scores

Beam search decoding: example

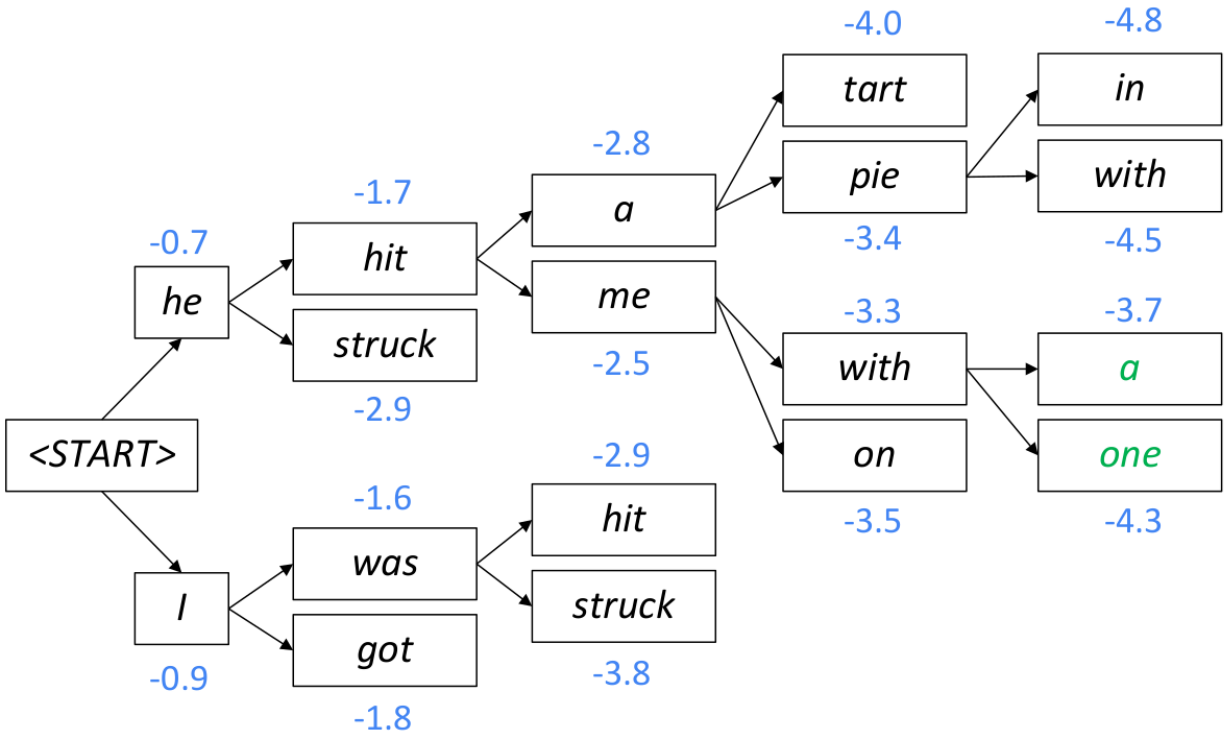
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

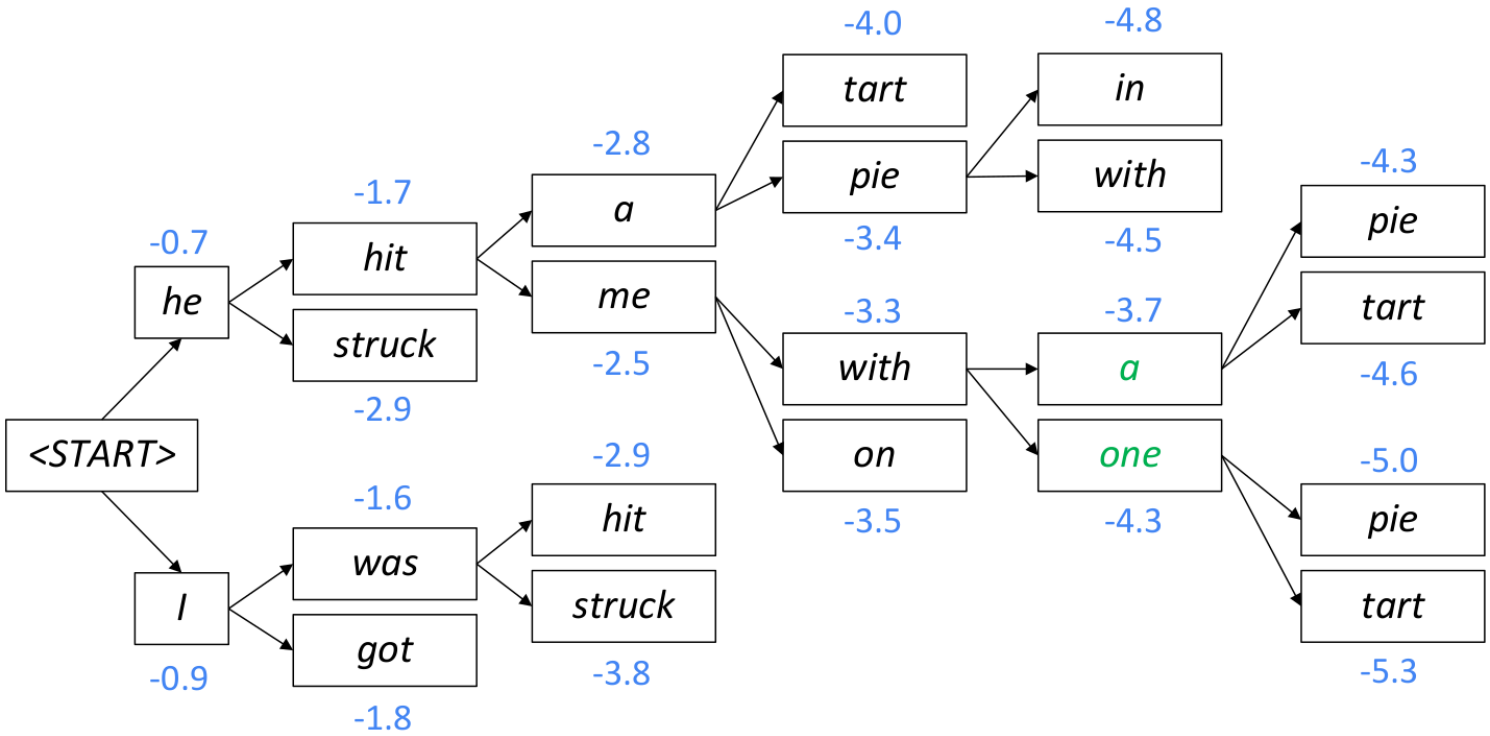
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Of these k^2 hypotheses, just keep k with highest scores

Beam search decoding: example

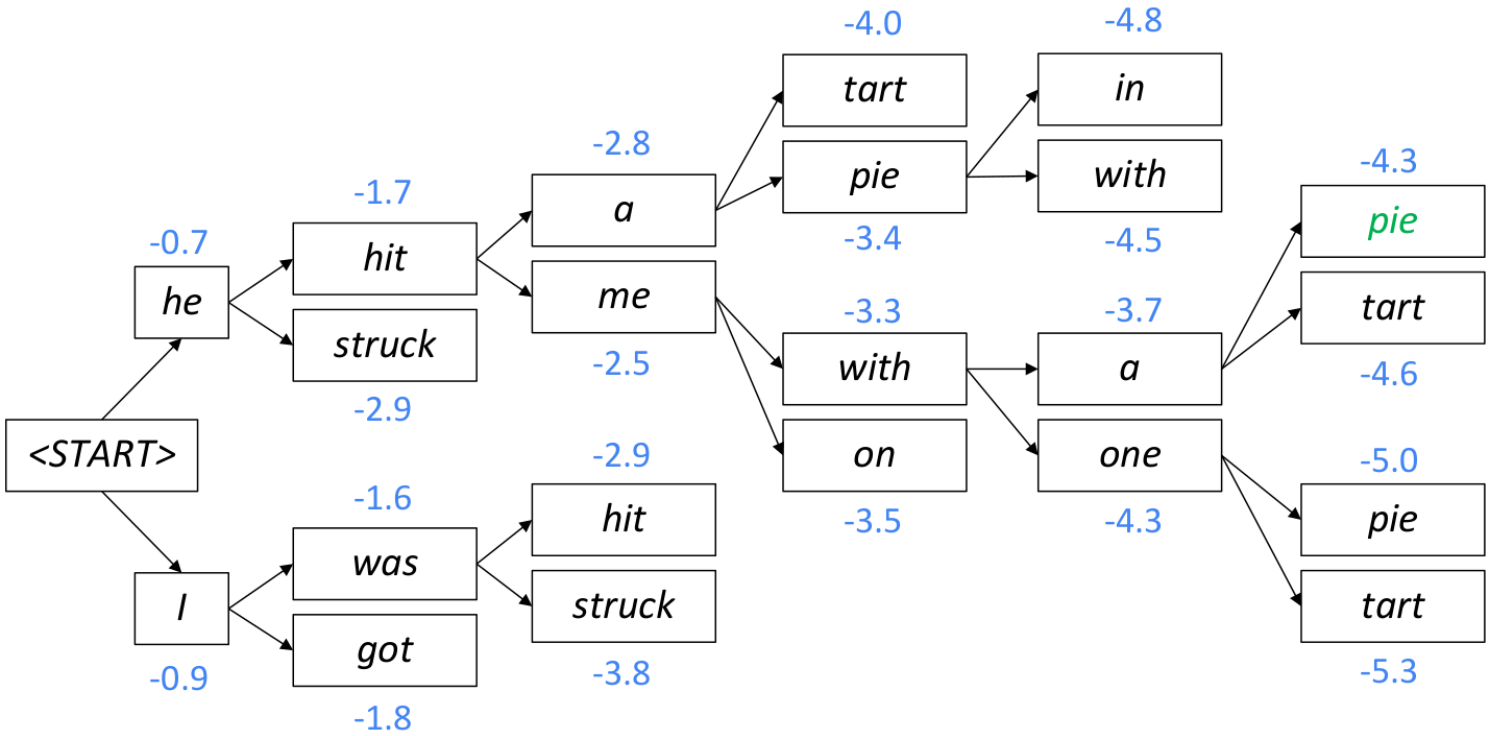
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding: example

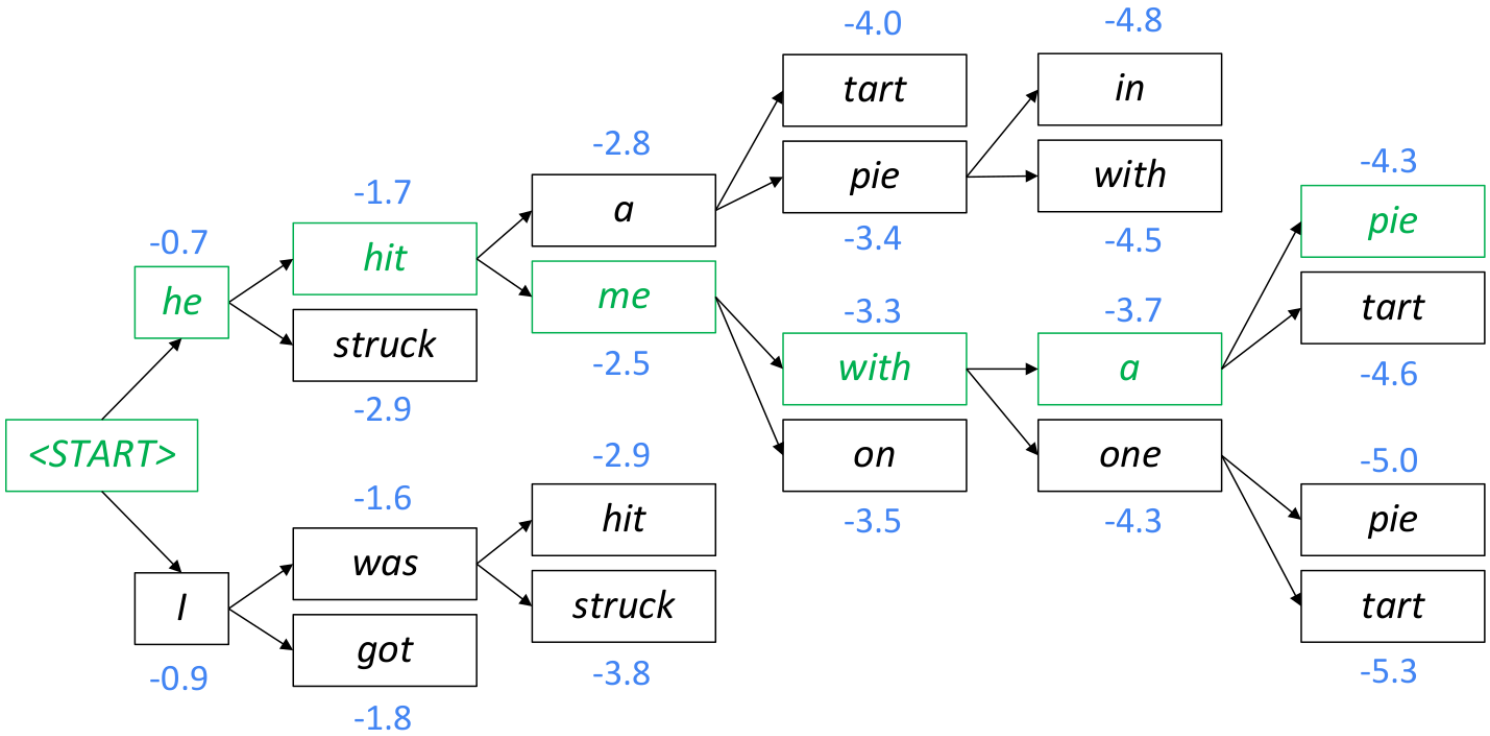
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



This is the top-scoring hypothesis!

Beam search decoding: example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Backtrack to obtain the full hypothesis

Beam search decoding: stopping criterion

- In greedy decoding, usually we decode until the model produces an <END> token
  - **For example:** *<START> he hit me with a pie <END>*

- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.

- Usually we continue beam search until:
  - We reach timestep T (where T is some pre-defined cutoff), or
  - We have at least n completed hypotheses (where n is pre-defined cutoff)

45

Beam search decoding: finishing up

- We have our list of completed hypotheses.
- How to select top one with highest score?

- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

- **Problem with this:** longer hypotheses have lower scores

- **Fix:** Normalize by length. Use this to select top one instead:

$$\frac{1}{t} \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

46

Beam search decoding: stopping criterion

- In greedy decoding, usually we decode until the model produces an <END> token
  - **For example:** *<START> he hit me with a pie <END>*

- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.

- Usually we continue beam search until:
  - We reach timestep T (where T is some pre-defined cutoff), or
  - We have at least n completed hypotheses (where n is pre-defined cutoff)

45

Przypomnienie: 4 poziomy modelu językowego

- **Poziom 0**: aplikacja
- **Poziom 1**: API generujące teksty
- **Poziom 2**: rozkład prawdopodobieństwa na tokenach
- **Poziom 3**: sieć neuronowa

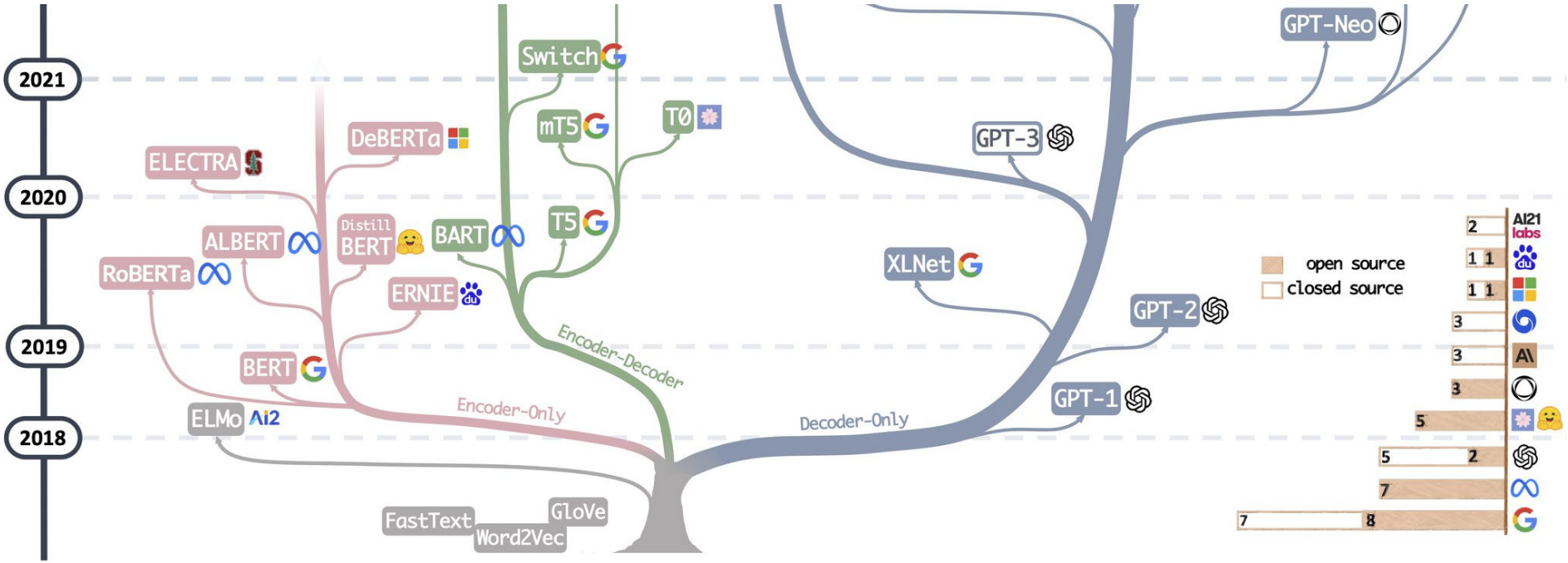
Poziom 3

Definicja

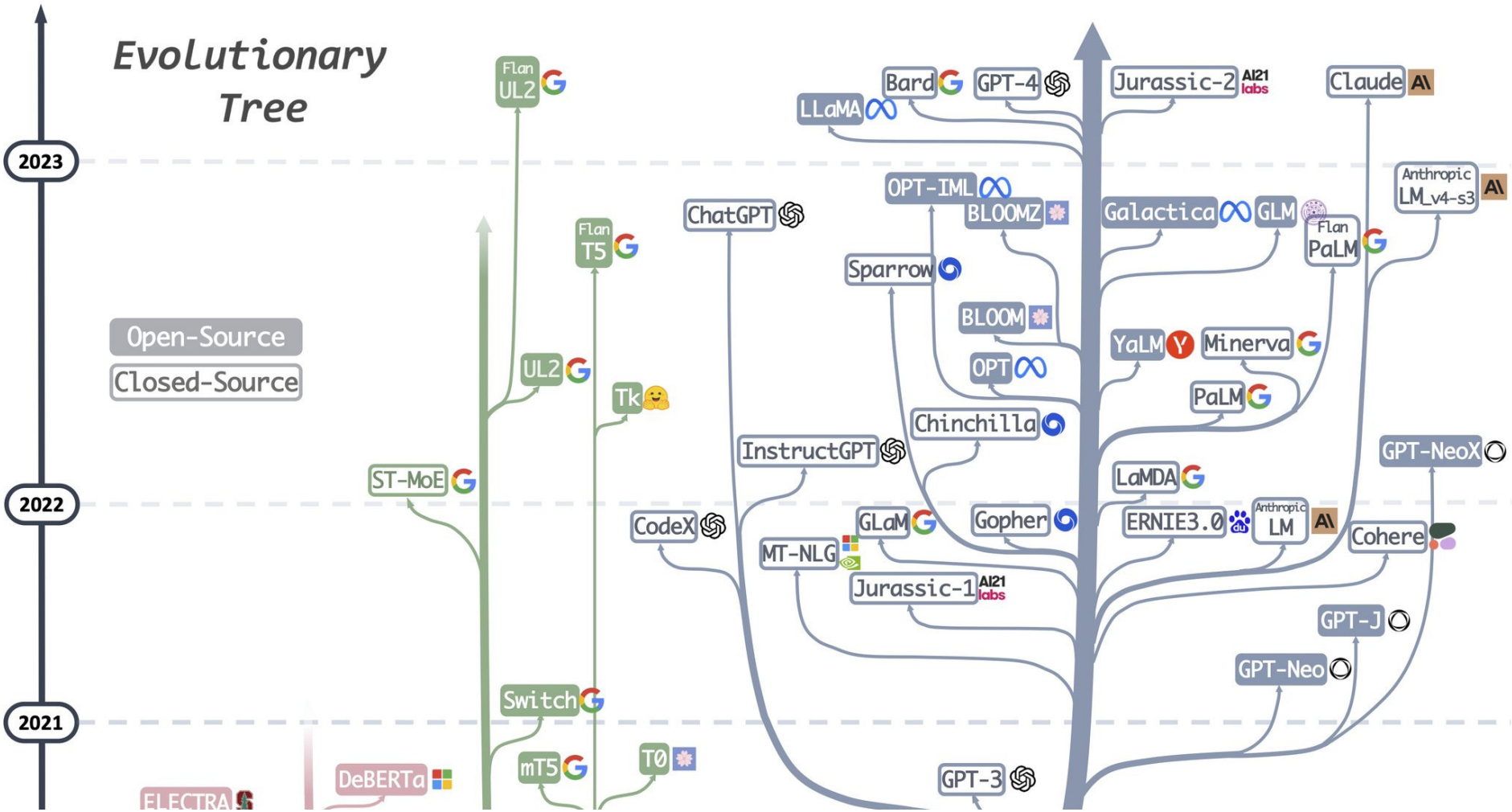
Model językowy (P3) jest siecią neuronową, która modeluje prawdopopodobieństwo następnego tokenu, pod warunkiem obecności prefiksu.

- Wejściem takiej sieci jest ciąg liczb naturalnych, będących numerami tokenów (od 1 do $|V|$, czyli wielkości słownika)
- Często taka sieć potrafi policzyć wiele rzeczy na raz: na przykład wszystkie prawwdopodobieństwa z poprzedniego slajdu (dla wielu zdań równocześnie)
- Obecnie absolutnym dominantem są tu sieci **Transformer**!

Drzewo genealogiczne modeli językowych



Drzewo genealogiczne modeli językowych



Sieci Transformer – silnik współczesnych modeli językowych

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

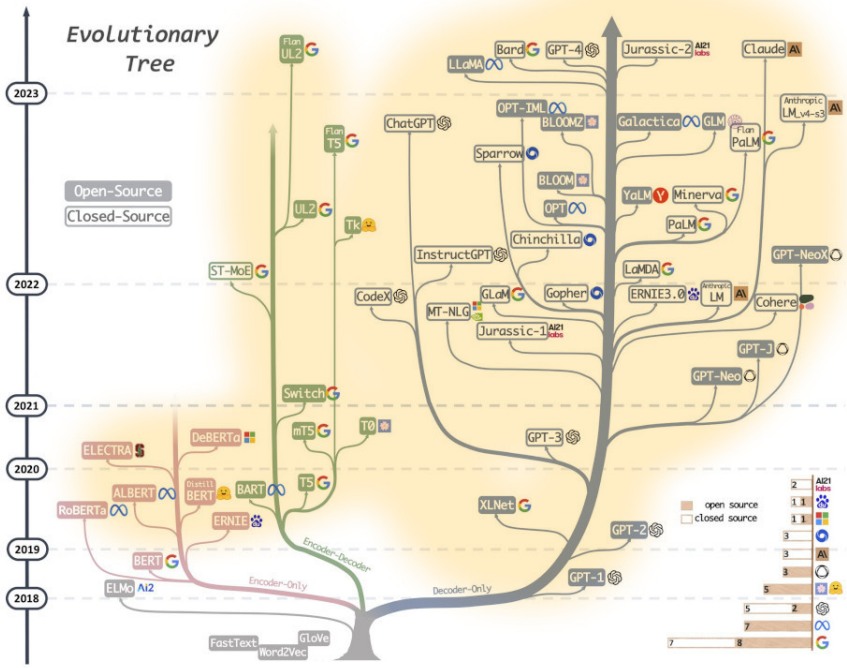
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions



Trening modelu

- W zadaniu next token prediction trenujemy model korzystając z (nieopisanego, dużego) korpusu
  - Liczenie statystyk n-gramowych, na bazie częstości wystąpień fraz w korpusie
  - Trening sieci neuronowych maksymalizuje dopasowanie modelu do danych
- Trening można przeprowadzać w wielu etapach, na różnych korpusach, przykładowo:
  - **Etap 1**: Trening wstępny (pretraining), na dużych korpusach tekstowych
  - **Etap 2**: Trening na danych dziedzinowych (mniejsze korpusy, bardziej „na temat")
  - **Etap 3**: Trening zadania docelowego (np klasyfikacji tekstów), ew. uczenie ze wzmocnieniem (jak mamy funkcję oceniającą generator)

Trening wstępny

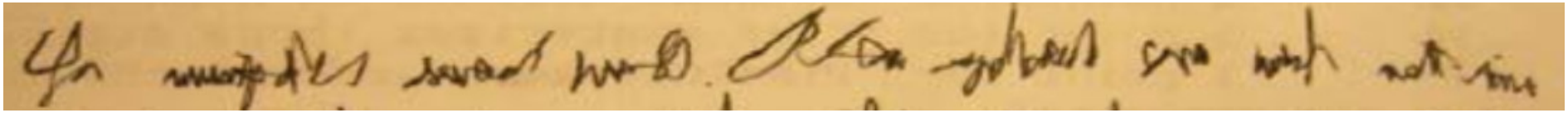
Czasem to jest wszystko, co potrzebujemy!

Przykładowe dziedziny, w których możliwe, że poprzestaniemy na treningu wstępnym:

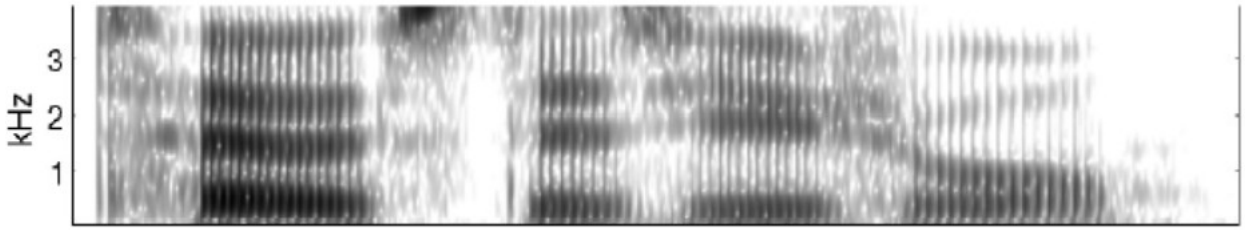
- Mały model tekstowy, dla konkretnego autora
  - (można też w takiej sytuacji teksty autora użyć dopiero w Etapie 2)
- Dane chemiczne: ciągi nukleotydów, ciągi aminokwasów (Nobel!)
- Dane muzyczne (pliki midi, pliki z nutami)
- Ciągi akcji lub ciągi stanów w grze (można tak wytrenować b.łatwo całkiem użytecznego agenta)
- Obrazki (długie lub kwadratowe, patrz następny slajd)

Dane graficzne

Długie obrazki, takie jak:



albo



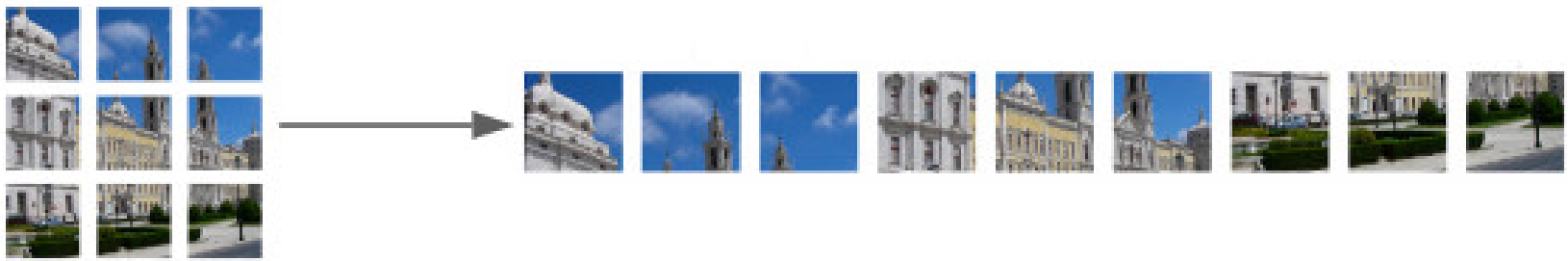
możemy podzielić na paski, które sklasteryzujemy i powiemy: **token = numer-klastra**

Uwaga

Problemem może być ciąg powtarzających się klastrów (dlaczego?). Możemy temu zaradzić, stosując **Run Length Encoding**. Zamieniamy: **aaaaabbccccccccc** na **a5 b2 c8**. Można też dodać tokeny takie jak **c10+**.

Dane graficzne (cd)

Jak obrazek jest bardziej kwadratowy, to nie jest problem, bo:



Uwaga

Takie modele mogą być użyte do rozpoznawania mowy, generowania podpisów, lub jako **część** systemu generującego dane określonego rodzaju (potrzeba bowiem jeszcze przejścia z ciągu tokenów do np. obrazka w wysokiej rozdzielczości)