

Transformery (część 2)

Paweł Rychlikowski

Instytut Informatyki UWr

17 grudnia 2025

Transformery. Gdzie jesteśmy?

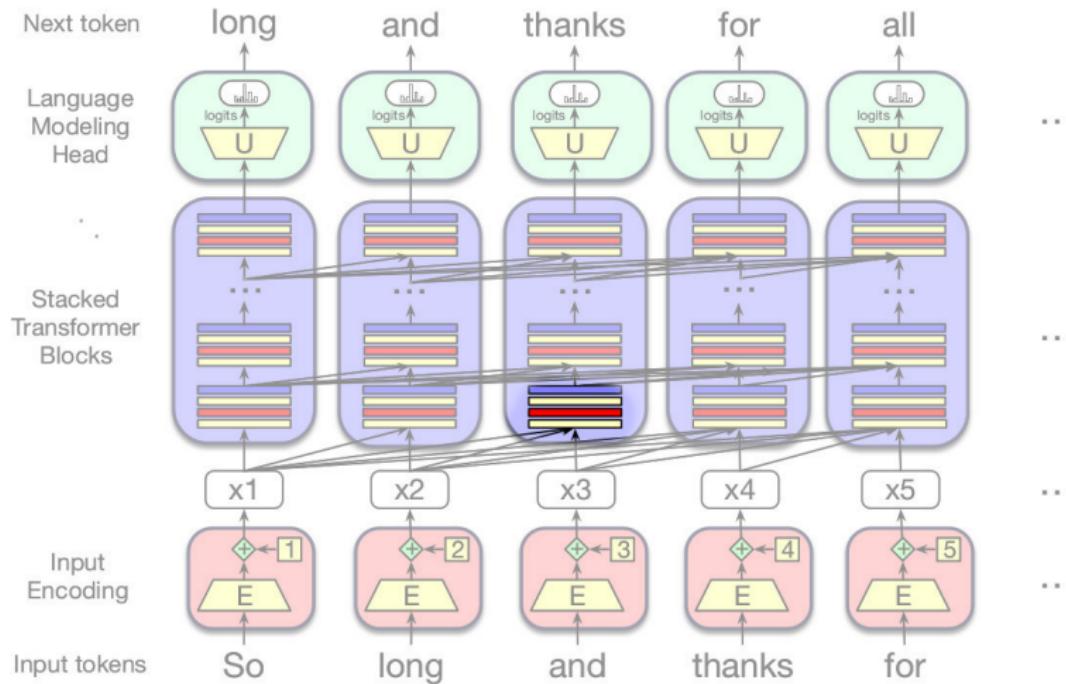


Figure 9.1 The architecture of a (left-to-right) transformer, showing how each input token get encoded, passed through a set of stacked transformer blocks, and then a language model head that predicts the next token.

Transformery

- Attention is all you need?

Transformery

- Attention is all you need? Really?

Transformery

- Attention is all you need? Really?
- Samo inteligentne uśrednianie to za mało – potrzebujemy klasycznej sieci neuronowej pomiędzy kolejnymi uśrednianiami (i paru drobiazgów do tego)

Transformery

- Attention is all you need? Really?
- Samo inteligentne uśrednianie to za mało – potrzebujemy klasycznej sieci neuronowej pomiędzy kolejnymi uśrednianiami (i paru drobiazgów do tego)
- Od teraz mówimy o operacjach na **pojedynczym** osadzeniu (co daje możliwość wykonywania równoległego)

Transformery

- Attention is all you need? Really?
- Samo inteligentne uśrednianie to za mało – potrzebujemy klasycznej sieci neuronowej pomiędzy kolejnymi uśrednianiami (i paru drobiazgów do tego)
- Od teraz mówimy o operacjach na **pojedynczym** osadzeniu (co daje możliwość wykonywania równoległego)

Wzór na sieć neuronową, przekształcającą osadzenie:

$$\text{FFN}(\mathbf{x}_i) = \text{ReLU}(\mathbf{x}_i \mathbf{W}_1 + b_1) \mathbf{W}_2 + b_2$$

$$(\text{ReLU}(x) = \max(x, 0))$$

Transformery. Normalizacja warstwy

$$\mu = \frac{1}{d} \sum_{i=1}^d x_i \quad (9.21)$$

$$\sigma = \sqrt{\frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2} \quad (9.22)$$

Given these values, the vector components are normalized by subtracting the mean from each and dividing by the standard deviation. The result of this computation is a new vector with zero mean and a standard deviation of one.

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma} \quad (9.23)$$

Finally, in the standard implementation of layer normalization, two learnable parameters, γ and β , representing gain and offset values, are introduced.

$$\text{LayerNorm}(\mathbf{x}) = \gamma \frac{(\mathbf{x} - \mu)}{\sigma} + \beta \quad (9.24)$$

Transformery. Normalizacja warstwy

$$\mu = \frac{1}{d} \sum_{i=1}^d x_i \quad (9.21)$$

$$\sigma = \sqrt{\frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2} \quad (9.22)$$

Given these values, the vector components are normalized by subtracting the mean from each and dividing by the standard deviation. The result of this computation is a new vector with zero mean and a standard deviation of one.

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma} \quad (9.23)$$

Finally, in the standard implementation of layer normalization, two learnable parameters, γ and β , representing gain and offset values, are introduced.

$$\text{LayerNorm}(\mathbf{x}) = \gamma \frac{(\mathbf{x} - \mu)}{\sigma} + \beta \quad (9.24)$$

Nie wyszła im ta nazwa! Normalizujemy pojedyncze osadzenie, nie warstwę.

Transformery. Schemat przepływów

- Myślimy o tym, że przez sieć płyną osadzenia, czasem się trochę mieszając z sąsiadami.

Transformery. Schemat przepływów

- Myślimy o tym, że przez sieć płyną osadzenia, czasem się trochę mieszając z sąsiadami.
- To zachowanie jest ok, nie chcemy go tracić, tylko lekko zmodyfikować (dodając pewne nowe rzeczy)

Transformery. Schemat przepływów

- Myślimy o tym, że przez sieć płyną osadzenia, czasem się trochę mieszając z sąsiadami.
- To zachowanie jest ok, nie chcemy go tracić, tylko lekko zmodyfikować (dodając pewne nowe rzeczy)

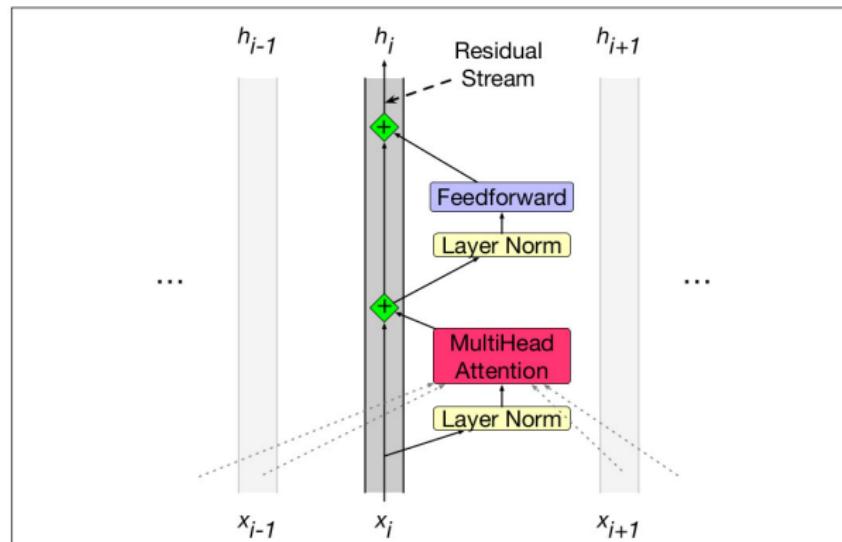


Figure 9.6 The architecture of a transformer block showing the **residual stream**. This figure shows the **prenorm** version of the architecture, in which the layer norms happen before the attention and feedforward layers rather than after.

Transformery. Definicja bloku

Putting it all together The function computed by a transformer block can be expressed by breaking it down with one equation for each component computation, using \mathbf{t} (of shape $[1 \times d]$) to stand for transformer and superscripts to demarcate each computation inside the block:

$$\mathbf{t}_i^1 = \text{LayerNorm}(\mathbf{x}_i) \quad (9.25)$$

$$\mathbf{t}_i^2 = \text{MultiHeadAttention}(\mathbf{t}_i^1, [\mathbf{x}_1^1, \dots, \mathbf{x}_N^1]) \quad (9.26)$$

$$\mathbf{t}_i^3 = \mathbf{t}_i^2 + \mathbf{x}_i \quad (9.27)$$

$$\mathbf{t}_i^4 = \text{LayerNorm}(\mathbf{t}_i^3) \quad (9.28)$$

$$\mathbf{t}_i^5 = \text{FFN}(\mathbf{t}_i^4) \quad (9.29)$$

$$\mathbf{h}_i = \mathbf{t}_i^5 + \mathbf{t}_i^3 \quad (9.30)$$

Transformery. Definicja bloku

Putting it all together The function computed by a transformer block can be expressed by breaking it down with one equation for each component computation, using \mathbf{t} (of shape $[1 \times d]$) to stand for transformer and superscripts to demarcate each computation inside the block:

$$\mathbf{t}_i^1 = \text{LayerNorm}(\mathbf{x}_i) \quad (9.25)$$

$$\mathbf{t}_i^2 = \text{MultiHeadAttention}(\mathbf{t}_i^1, [\mathbf{x}_1^1, \dots, \mathbf{x}_N^1]) \quad (9.26)$$

$$\mathbf{t}_i^3 = \mathbf{t}_i^2 + \mathbf{x}_i \quad (9.27)$$

$$\mathbf{t}_i^4 = \text{LayerNorm}(\mathbf{t}_i^3) \quad (9.28)$$

$$\mathbf{t}_i^5 = \text{FFN}(\mathbf{t}_i^4) \quad (9.29)$$

$$\mathbf{h}_i = \mathbf{t}_i^5 + \mathbf{t}_i^3 \quad (9.30)$$

- Te obliczenia są powtarzane kilka/kilkadziesiąt razy (dla różnych parametrów, choć były i warianty zachowujące te same parametry)

Gdzie jesteśmy?

Na poprzednim slajdzie były **pełne obliczenia** wyznaczające kontekstowe osadzenia. Zostało nam jeszcze:

- Omówić wykorzystanie tych osadzeń do obliczania prawdopodobieństw.
- Omówić kwestie równoległości obliczeń i wsadów.
- Omówić trening transformerów (nieco dokładniej, niż od tej pory)

Głowa językowa (language modeling head)

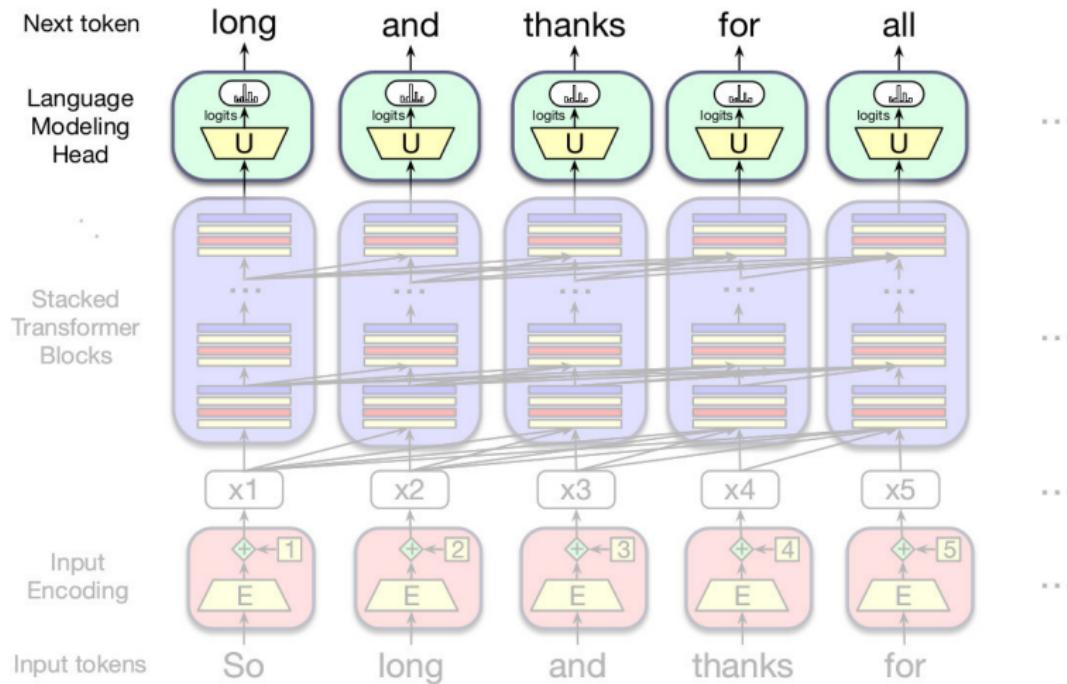


Figure 9.1 The architecture of a (left-to-right) transformer, showing how each input token get encoded, passed through a set of stacked transformer blocks, and then a language model head that predicts the next token.

Głowa językowa (language modeling head)

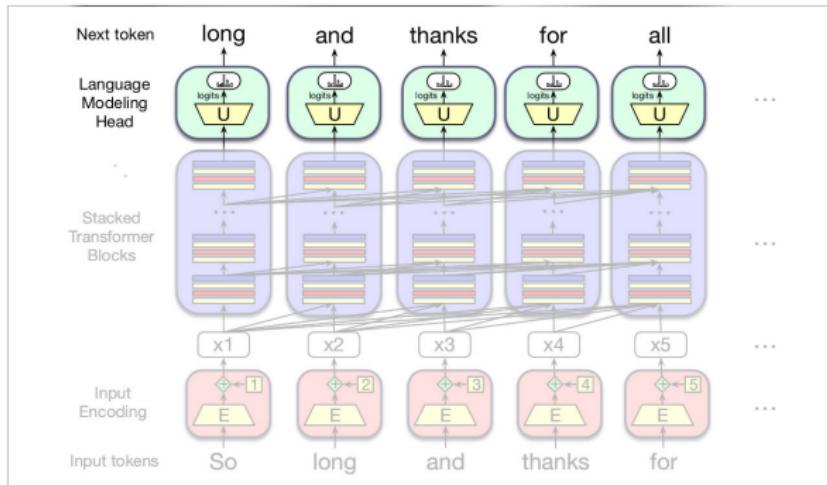


Figure 9.1 The architecture of a (left-to-right) transformer, showing how each input token get encoded, passed through a set of stacked transformer blocks, and then a language model head that predicts the next token.

- Osadzenia mnożymy przez macierz U , o wymiarach $d \times V$ (czyli otrzymamy punkty dla każdego tokenu w słowniku)
- Na to nakładamy operację **softmax**

Głowa językowa (language modeling head)

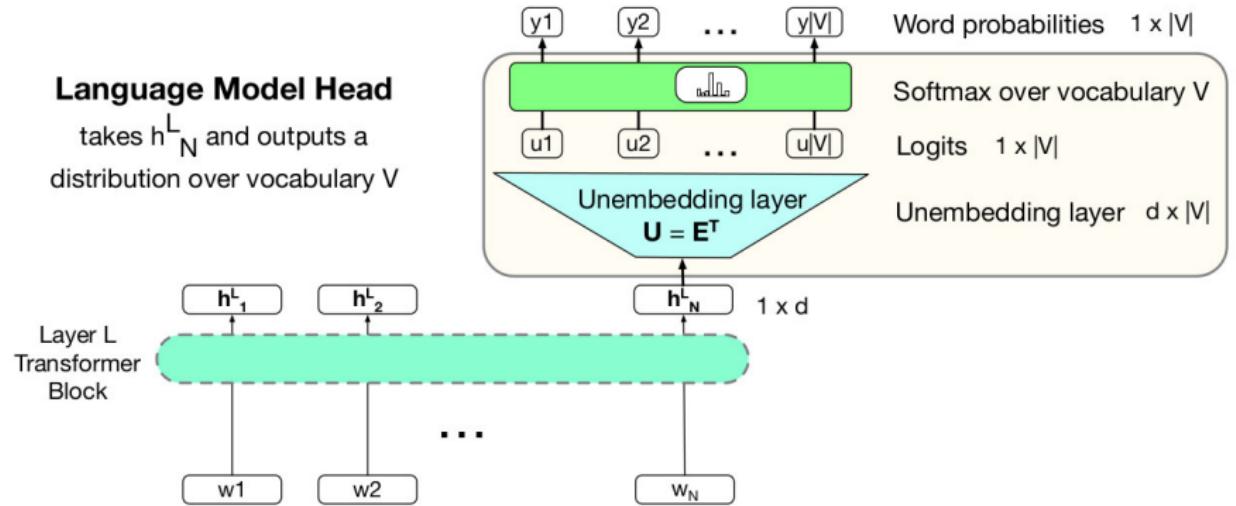
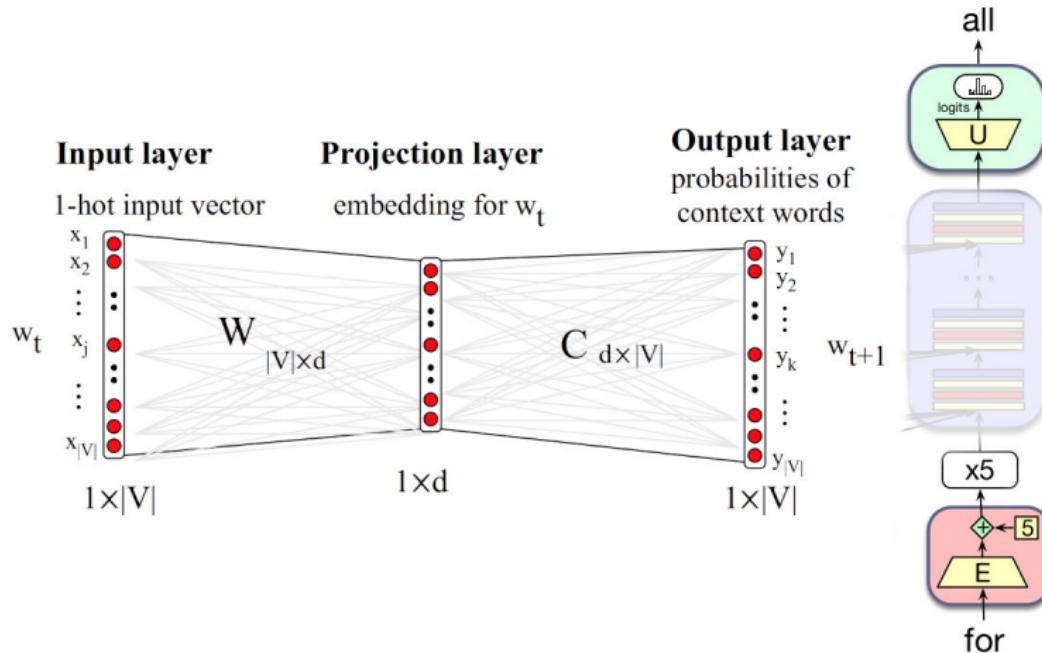


Figure 9.14 The language modeling head: the circuit at the top of a transformer that maps from the output embedding for token N from the last transformer layer (h_N^L) to a probability distribution over words in the vocabulary V .

- E^T występuje dlatego, że macierz U jest zorientowana kolumnowo (a nie wierszowo, jak tradycyjne osadzenia)
- **Logity** – to słowo oznacza wartości sieci przekazywane softmaxowi.

Transformer vs word2vec



Transformer dokłada *trochę* obliczeń do bigramowego word2vec-a (ale w rezydualnych potokach)

Atencja w wersji macierzowej

- Chcemy obsłużyć wszystkie $x_i, i \in \{1, \dots, n\}$ naraz, za pomocą pojedynczych mnożeń macierzy
- Umieścimy je w tym celu w macierzy X , jako n wierszy.

Atencja w wersji macierzowej

- Chcemy obsłużyć wszystkie $x_i, i \in \{1, \dots, n\}$ naraz, za pomocą pojedynczych mnożeń macierzy
- Umieścimy je w tym celu w macierzy X , jako n wierszy.

$$\mathbf{Q} = \mathbf{XW}^Q; \quad \mathbf{K} = \mathbf{XW}^K; \quad \mathbf{V} = \mathbf{XW}^V \quad (9.31)$$

Given these matrices we can compute all the requisite query-key comparisons simultaneously by multiplying \mathbf{Q} and \mathbf{K}^T in a single matrix multiplication. The product is of shape $N \times N$, visualized in Fig. 9.8.

| | | N | | | |
|---|-------|-------|-------|-------|-------|
| | | q1·k1 | q1·k2 | q1·k3 | q1·k4 |
| N | q2·k1 | q2·k2 | q2·k3 | q2·k4 | |
| | q3·k1 | q3·k2 | q3·k3 | q3·k4 | |
| | q4·k1 | q4·k2 | q4·k3 | q4·k4 | |
| | | | | | |

Figure 9.8 The $N \times N$ $\mathbf{Q}\mathbf{K}^T$ matrix showing how it computes all $q_i \cdot k_j$ comparisons in a single matrix multiple.

Atencja w wersji macierzowej z maską

- Poniżej zobaczymy pełen wzór na atencję, w wersji macierzowej
- Operator **mask** jest identycznością dla BERT-a, a dla GPT wstawia $-\infty$ w miejscach niedozwolonej atencji (czyli patrzącej w przód)

Atencja w wersji macierzowej z maską

- Poniżej zobaczymy pełen wzór na atencję, w wersji macierzowej
- Operator **mask** jest identycznością dla BERT-a, a dla GPT wstawia $-\infty$ w miejscach niedozwolonej atencji (czyli patrzącej w przód)

$$\mathbf{A} = \text{softmax} \left(\text{mask} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \right) \right) \mathbf{V} \quad (11.1)$$

The figure consists of two tables labeled (a) and (b), each representing an $N \times N$ matrix. The matrices show the product of query vectors (q_i) and key vectors (k_j). The diagonal elements are positive values, while the upper-triangle portion of both matrices has values set to $-\infty$.

| | | | |
|-----------------|-----------------|-----------------|-----------------|
| $q_1 \cdot k_1$ | $-\infty$ | $-\infty$ | $-\infty$ |
| $q_2 \cdot k_1$ | $q_2 \cdot k_2$ | $-\infty$ | $-\infty$ |
| $q_3 \cdot k_1$ | $q_3 \cdot k_2$ | $q_3 \cdot k_3$ | $-\infty$ |
| $q_4 \cdot k_1$ | $q_4 \cdot k_2$ | $q_4 \cdot k_3$ | $q_4 \cdot k_4$ |

N

(a)

| | | | |
|-----------------|-----------------|-----------------|-----------------|
| $q_1 \cdot k_1$ | $q_1 \cdot k_2$ | $q_1 \cdot k_3$ | $q_1 \cdot k_4$ |
| $q_2 \cdot k_1$ | $q_2 \cdot k_2$ | $q_2 \cdot k_3$ | $q_2 \cdot k_4$ |
| $q_3 \cdot k_1$ | $q_3 \cdot k_2$ | $q_3 \cdot k_3$ | $q_3 \cdot k_4$ |
| $q_4 \cdot k_1$ | $q_4 \cdot k_2$ | $q_4 \cdot k_3$ | $q_4 \cdot k_4$ |

N

(b)

Figure 11.2 The $N \times N$ $\mathbf{Q}\mathbf{K}^T$ matrix showing the $q_i \cdot k_j$ values, with the upper-triangle portion of the comparisons matrix zeroed out (set to $-\infty$, which the softmax will turn to zero).

Trening sieci neuronowych z lotu ptaka

- Sieć neuronowa jest **różniczkowalną** funkcją $f_\theta : \mathcal{R}^n \rightarrow \mathcal{R}^m$, gdzie θ to parametry

Trening sieci neuronowych z lotu ptaka

- Sieć neuronowa jest **różniczkowalną** funkcją $f_\theta : \mathcal{R}^n \rightarrow \mathcal{R}^m$, gdzie θ to parametry
- **Trening sieci** to dostosowanie parametrów θ do zestawu danych uczących, czyli

$$\{(x_i, y_i)\}_{i=1}^N, \text{ gdzie } x_i \in \mathcal{R}^n, y_i \in \mathcal{R}^m$$

Trening sieci neuronowych z lotu ptaka

- Sieć neuronowa jest **różniczkowalną** funkcją $f_{\theta} : \mathcal{R}^n \rightarrow \mathcal{R}^m$, gdzie θ to parametry
- **Trening sieci** to dostosowanie parametrów θ do zestawu danych uczących, czyli

$$\{(x_i, y_i)\}_{i=1}^N, \text{ gdzie } x_i \in \mathcal{R}^n, y_i \in \mathcal{R}^m$$

- Trening jest minimalizacją **funkcji kosztu (loss)**, która bierze sieć, dane uczące i zwraca nieujemną liczbę rzeczywistą (określającą, jak bardzo funkcja **nie pasuje** do danych)

Trening sieci neuronowych z lotu ptaka

- Sieć neuronowa jest **różniczkowalną** funkcją $f_\theta : \mathcal{R}^n \rightarrow \mathcal{R}^m$, gdzie θ to parametry
- **Trening sieci** to dostosowanie parametrów θ do zestawu danych uczących, czyli

$$\{(x_i, y_i)\}_{i=1}^N, \text{ gdzie } x_i \in \mathcal{R}^n, y_i \in \mathcal{R}^m$$

- Trening jest minimalizacją **funkcji kosztu (loss)**, która bierze sieć, dane uczące i zwraca nieujemną liczbę rzeczywistą (określającą, jak bardzo funkcja **nie pasuje** do danych)

Procedura minimalizacji korzysta z gradientu, zmieniając parametry θ w kierunku największego spadku *kosztu*.

GPT. Trening

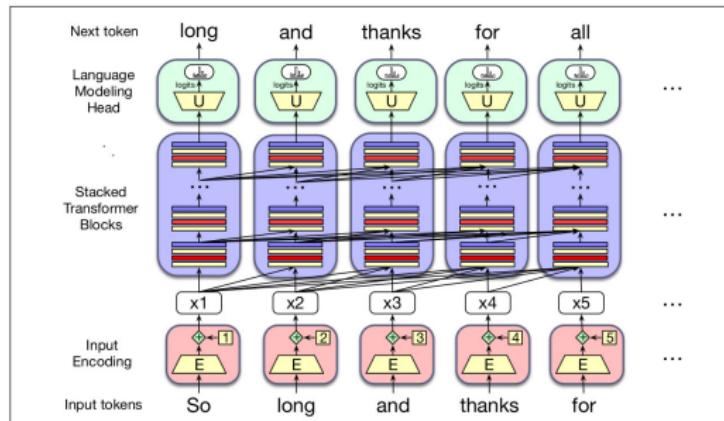


Figure 9.1 The architecture of a (left-to-right) transformer, showing how each input token get encoded, passed through a set of stacked transformer blocks, and then a language model head that predicts the next token.

Koszt jest sumą **wszystkich** logarytmów prawdopodobieństw, dla wszystkich słów **przewidywanych** (czyli przesuniętego o 1 ciągu wejściowego)

W tym przypadku mamy ...

$$-\log(\text{long}) + -\log(\text{and}) + -\log(\text{thanks}) + \dots$$

BERT. Trening (1)

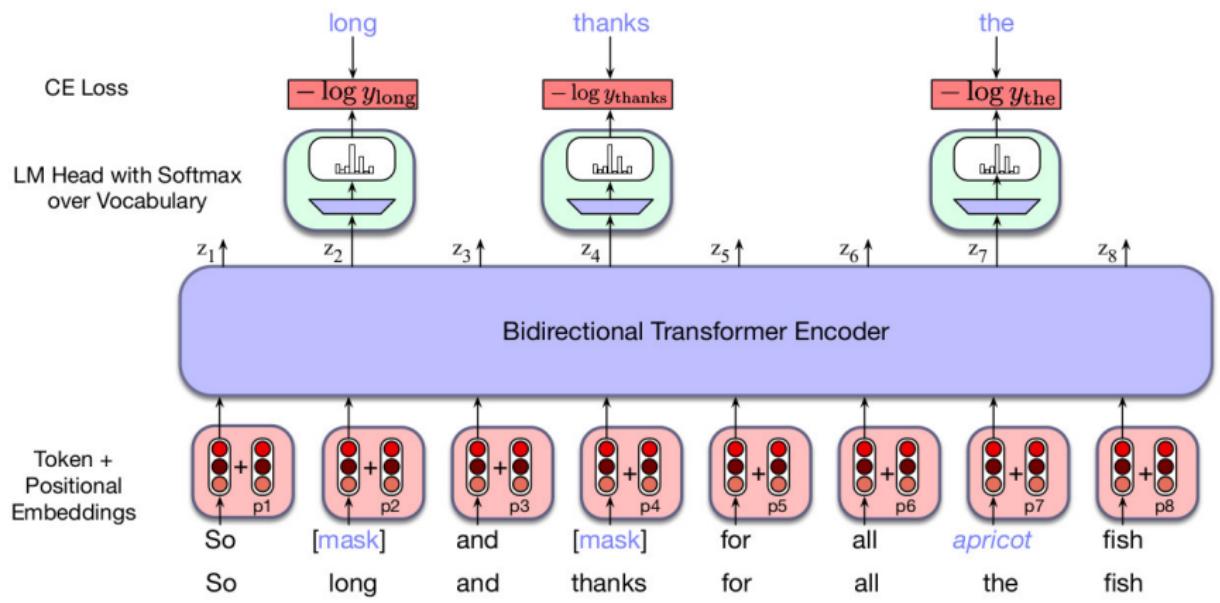


Figure 11.3 Masked language model training. In this example, three of the input tokens are selected, two of which are masked and the third is replaced with an unrelated word. The probabilities assigned by the model to these three items are used as the training loss. The other 5 tokens don't play a role in training loss.

BERT. Trening (1)

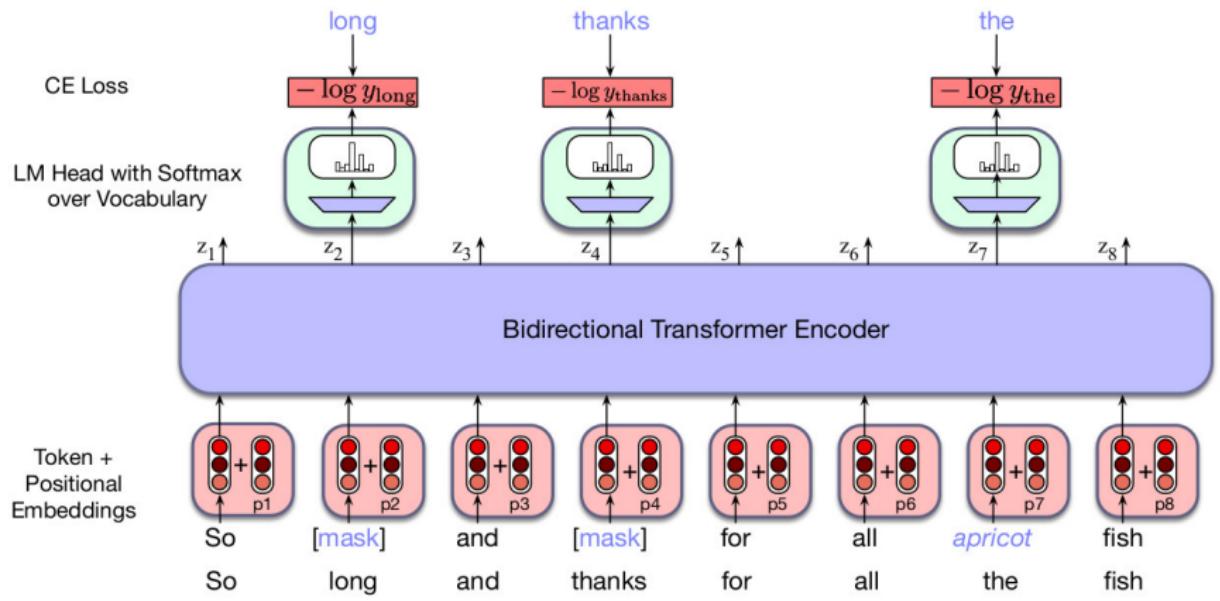


Figure 11.3 Masked language model training. In this example, three of the input tokens are selected, two of which are masked and the third is replaced with an unrelated word. The probabilities assigned by the model to these three items are used as the training loss. The other 5 tokens don't play a role in training loss.

Aby otrzymać Loss sumujemy wszystkie logarytmy, liczone na wyróżninych pozycjach.

BERT. Trening (2)

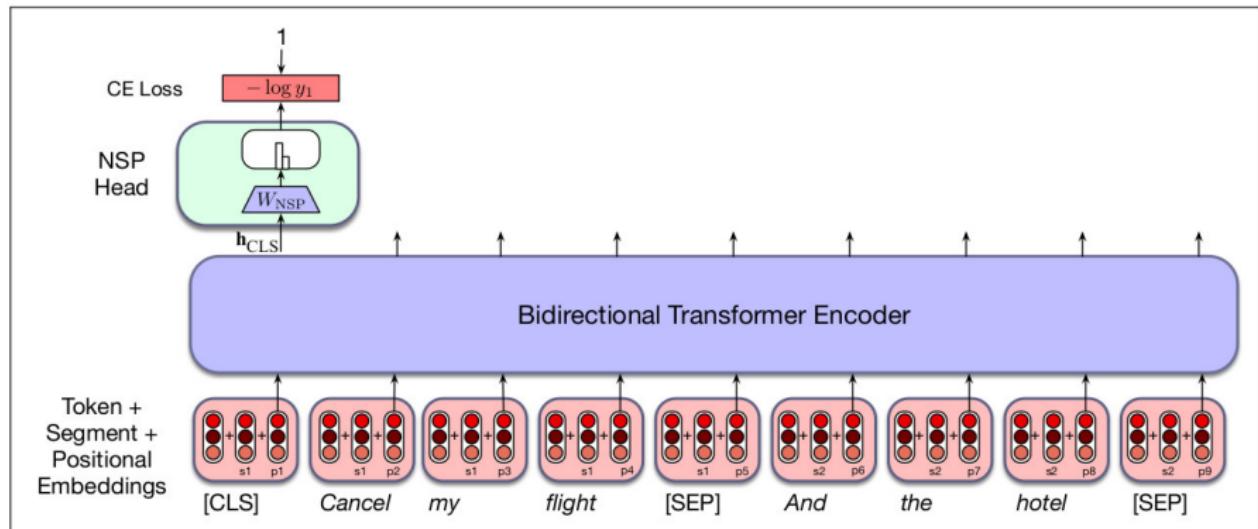


Figure 11.4 An example of the NSP loss calculation.

Do kosztu z poprzedniego slajdu dodajemy (być może z jakąś wagą) koszt związany z klasyfikacją (pary zdań)

Koder-dekoder

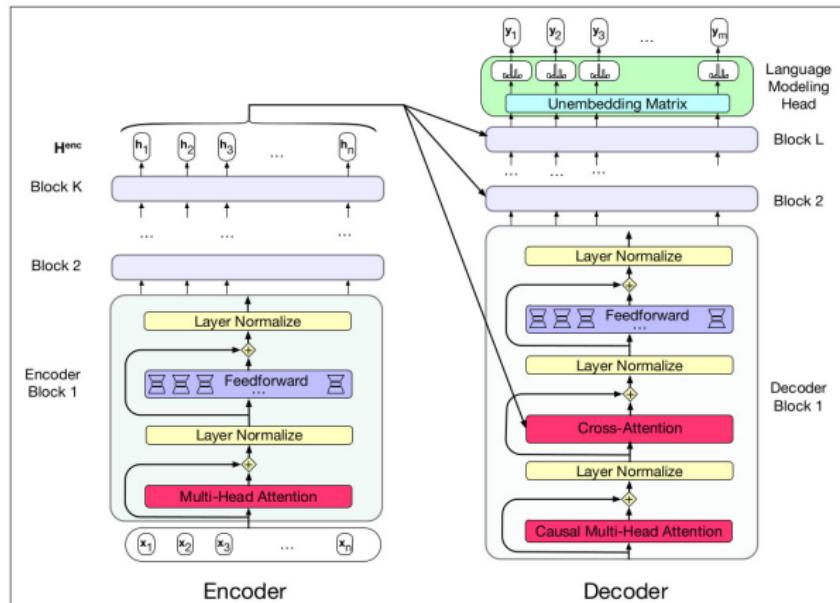


Figure 13.6 The transformer block for the encoder and the decoder. The final output of the encoder $\mathbf{H}^{enc} = \mathbf{h}_1, \dots, \mathbf{h}_n$ is the context used in the decoder. The decoder is a standard transformer except with one extra layer, the **cross-attention** layer, which takes that encoder output \mathbf{H}^{enc} and uses it to form its \mathbf{K} and \mathbf{V} inputs.

- Część kodera nie ma bezpośredniego wpływu na funkcję kosztu!
- self-attention vs cross-attention (ta druga jest pomiędzy dwiema osobnymi sieciami), nie musi patrzeć na tym samym poziomie.
- Koder może mieć atencję BERT-like

Trening wstępny i dostrajanie

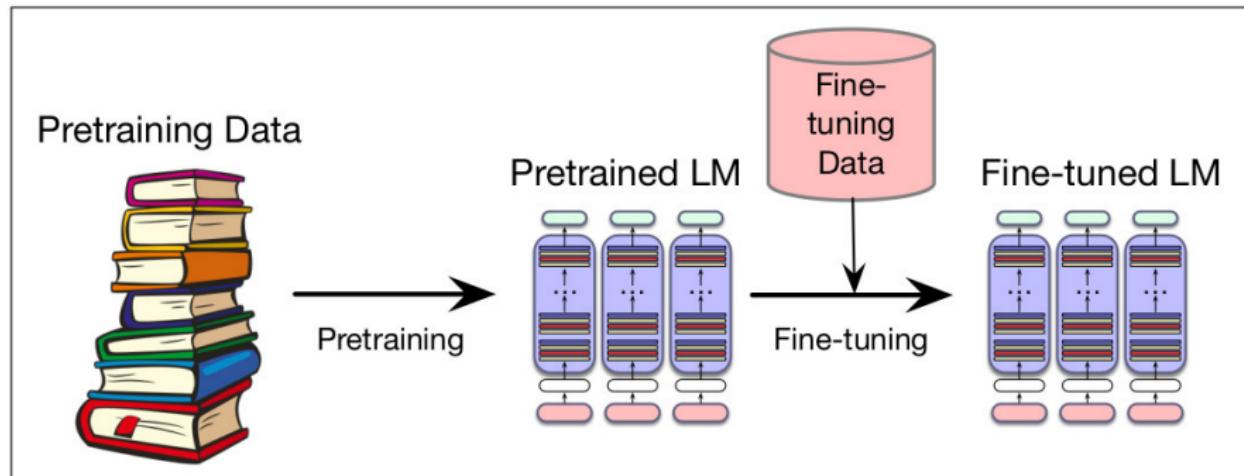


Figure 10.6 Pretraining and finetuning. A pre-trained model can be finetuned to a particular domain, dataset, or task. There are many different ways to finetune, depending on exactly which parameters are updated from the finetuning data: all the parameters, some of the parameters, or only the parameters of specific extra circuitry.

Trening wstępny i dostrajanie

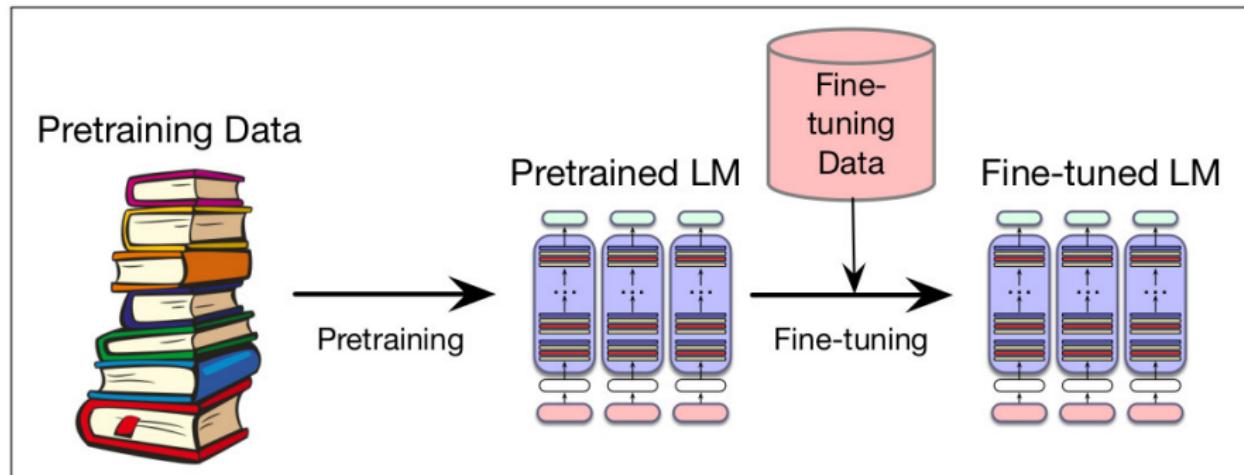


Figure 10.6 Pretraining and finetuning. A pre-trained model can be finetuned to a particular domain, dataset, or task. There are many different ways to finetune, depending on exactly which parameters are updated from the finetuning data: all the parameters, some of the parameters, or only the parameters of specific extra circuitry.

Być może dokładnie ta sama procedura trenująca, tylko inne dane
(ewentualnie większa stała ucząca, dla szybszego treningu)

Różne rodzaje dostrajania

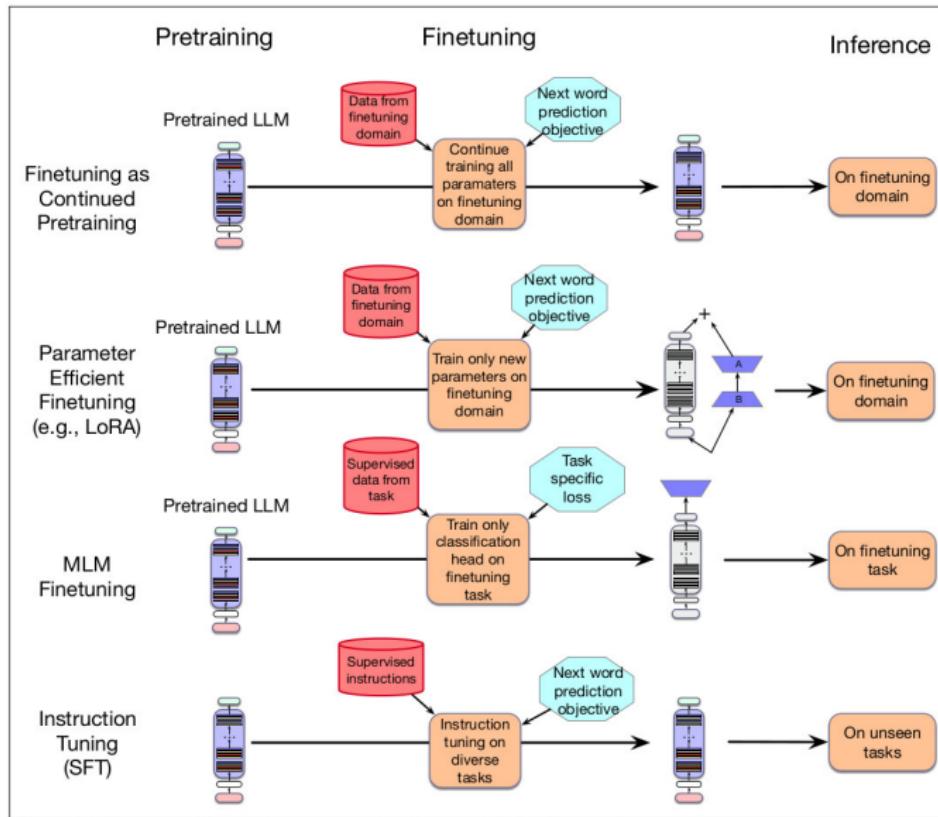


Figure 12.4 Instruction tuning compared to the other kinds of finetuning.

Uczenie ze wzmacnianiem w transformerach

- Założymy, że mamy jakąś zewnętrzną funkcję akceptującą/odrzucającą wynik generacji (pisaliśmy takie funkcje na pracowni)

Uczenie ze wzmacnianiem w transformerach

- Założymy, że mamy jakąś zewnętrzną funkcję akceptującą/odrzucającą wynik generacji (pisaliśmy takie funkcje na pracowni)
- Taka funkcja może być też wytrenowana

Uczenie ze wzmacnianiem w transformerach

- Założymy, że mamy jakąś zewnętrzną funkcję akceptującą/odrzucającą wynik generacji (pisaliśmy takie funkcje na pracowni)
- Taka funkcja może być też wytrenowana
 - ▶ Na przykład dostrajamy model, by po każdym końcu zdania wygenerował jeden z dwóch tekstów: **[+1]** albo **[-1]**, mówiących o tym, czy model zrobił wszystko jak trzeba.

Uczenie ze wzmacnianiem w transformerach

- Założymy, że mamy jakąś zewnętrzną funkcję akceptującą/odrzucającą wynik generacji (pisaliśmy takie funkcje na pracowni)
- Taka funkcja może być też wytrenowana
 - ▶ Na przykład dostrajamy model, by po każdym końcu zdania wygenerował jeden z dwóch tekstów: **[+1]** albo **[-1]**, mówiących o tym, czy model zrobił wszystko jak trzeba.

Definicja

Trening, w którym **oceniacz** jest wytrenowany na podstawie ludzkich preferencji nazywamy jest **Reinforcement Learning with Human Feedback (RLHF)**

Uczenie ze wzmacnianiem w transformerach

Trening

- Generujemy zdanie/kawałek tekstu
- Oceniamy zewnętrznym ewaluatorem

Trening

- Generujemy zdanie/kawałek tekstu
- Oceniamy zewnętrznym ewaluatorem
- Czy jest ok?
 - ▶ **Tak:** zwiększamy prawdopodobieństwo zdania (dokładnie tak, jak po zaobserwowaniu w korpusie)
 - ▶ **Nie:** zmniejszamy prawdopodobieństwo zdania (tak, jak poprzednio, ale koszt jest pomnożony przez -1)

Trening

- Generujemy zdanie/kawałek tekstu
- Oceniamy zewnętrznym ewaluatorem
- Czy jest ok?
 - ▶ **Tak:** zwiększamy prawdopodobieństwo zdania (dokładnie tak, jak po zaobserwowaniu w korpusie)
 - ▶ **Nie:** zmniejszamy prawdopodobieństwo zdania (tak, jak poprzednio, ale koszt jest pomnożony przez -1)

Oczywiście to potwarzamy tysiące/miliony razy.

Uczenie ze wzmacnianiem w transformerach

Trening

- Generujemy zdanie/kawałek tekstu
- Oceniamy zewnętrznym ewaluatorem
- Czy jest ok?
 - ▶ **Tak:** zwiększamy prawdopodobieństwo zdania (dokładnie tak, jak po zaobserwowaniu w korpusie)
 - ▶ **Nie:** zmniejszamy prawdopodobieństwo zdania (tak, jak poprzednio, ale koszt jest pomnożony przez -1)

Oczywiście to potwarzamy tysiące/miliony razy.

Uwaga

Warto zadbać, żeby przykłady pozytywne i negatywne były mniej więcej w równowadze

Trening ChataGPT

- Kolejne slajdy będą z wystąpienia A. Karpathy'ego z OpenAI
- Link: <https://www.youtube.com/watch?v=bZQun8Y4L2A>

Trening ChataGPT

- Kolejne slajdy będą z wystąpienia A. Karpathy'ego z OpenAI
- Link: <https://www.youtube.com/watch?v=bZQun8Y4L2A>

Implementacje GPT

Później będziemy też używać dwóch implementacji GPT Karpathy'ego:

- **miniGPT** (Link: <https://github.com/karpathy/minGPT>)
- **nanoGPT** (Link: <https://github.com/karpathy/nanoGPT>)

Base models are NOT 'Assistants'

- Base model does not answer questions
- It only wants to complete internet documents
- Often responds to questions with more questions, etc.:

Write a poem about bread and cheese.

Write a poem about someone who died of starvation.

Write a poem about angel food cake.

Write a poem about someone who choked on a ham sandwich.

Write a poem about a hostess who makes the

It can be tricked into performing tasks with prompt engineering:

Here is a poem about bread and cheese:

Bread and cheese is my desire,

And it shall be my destiny.

Bread and cheese is my desire,

And it shall be my destiny.

Here is a poem about cheese:

|

Base models are NOT 'Assistants'

(They can be somewhat tricked
into being AI assistants)

Make it look like document

Few-shot prompt

Insert query here →

Completion

The following is a conversation between a Human and a helpful, honest and harmless AI Assistant.

[Human]
Hi, how are you?

[Assistant]
I'm great, thank you for asking. How I can help you today?

[Human]
I'd like to know what is $2+2$ thanks

[Assistant]
 $2+2$ is 4.

[Human]
Great job.

[Assistant]
What else can I help you with?

[Human]
What is the capital of France?

[Assistant]
Paris.

GPT Assistant training pipeline



Transformery wnioskujące. Brudnopsy

SHOW YOUR WORK: SCRATCHPADS FOR INTERMEDIATE COMPUTATION WITH LANGUAGE MODELS

Maxwell Nye^{1,2*} Anders Johan Andreassen³ Guy Gur-Ari³ Henryk Michalewski²

Jacob Austin² David Bieber² David Dohan² Aitor Lewkowycz³ Maarten Bosma²

David Luan²

Charles Sutton²

Augustus Odena²

¹MIT

²Google Research, Brain Team

³Google Research, Blueshift Team

ABSTRACT

Large pre-trained language models perform remarkably well on tasks that can be done “in one pass”, such as generating realistic text (Brown et al., 2020) or synthesizing computer programs (Chen et al., 2021; Austin et al., 2021). However, they struggle with tasks that require unbounded multi-step computation, such as adding integers (Brown et al., 2020) or *executing* programs (Austin et al., 2021). Surprisingly, we find that these same models are able to perform complex multi-step computations—even in the few-shot regime—when asked to perform the operation “step by step”, showing the results of intermediate computations. In particular, we train Transformers to perform multi-step computations by asking them to emit intermediate computation steps into a “scratchpad”. On a series of increasingly complex tasks ranging from long addition to the execution of arbitrary

Transformery wnioskujące. Brudnopsy

Input:
2 9 + 5 7

Target:
<scratch>
2 9 + 5 7 , C: 0
2 + 5 , 6 C: 1 # added 9 + 7 = 6 carry 1
, 8 6 C: 0 # added 2 + 5 + 1 = 8 carry 0
0 8 6
</scratch>
8 6

Figure 2: Example of input and target for addition with a scratchpad. The carry is recorded in the digit following “C.”. Comments (marked by #) are added for clarity and are not part of the target.

Dodawanie możemy rozpisać cyfra po cyfrze, z przeniesieniem.

Transformery z brudnopsisem. Skuteczność dodawania

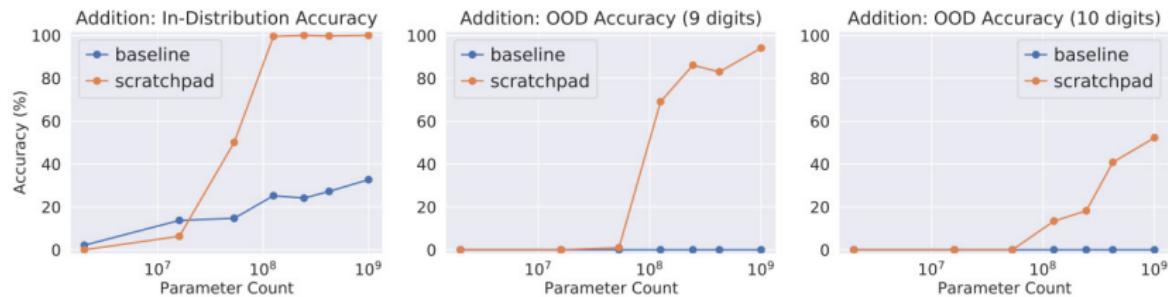


Figure 3: Using a scratchpad significantly improves the performance of pre-trained Transformer-based models on addition, including their ability to generalize out of the training distribution to numbers with more digits. Models were trained on 1-8 digit addition. The baseline models were trained without intermediate scratchpad steps.

Używane były tu modele wstępnie wytrenowane na stronach WWW (korpusach „ogólnych”), które następnie dotrenowywano na danych z brudnopsisem lub bez.

Transformery z brudnopsisem. Skuteczność dodawania

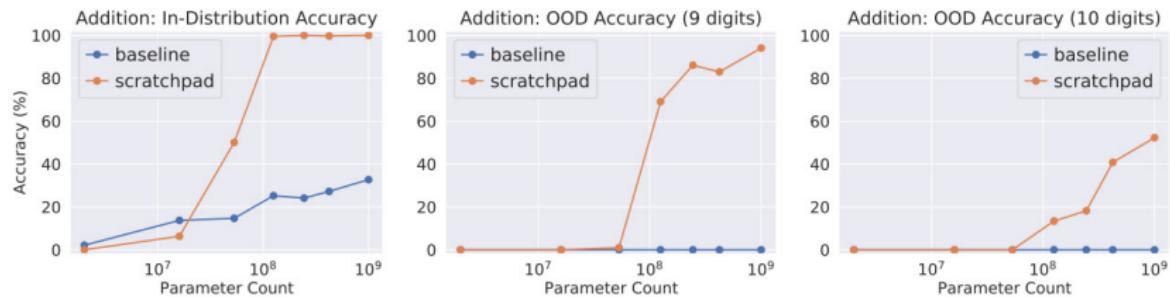


Figure 3: Using a scratchpad significantly improves the performance of pre-trained Transformer-based models on addition, including their ability to generalize out of the training distribution to numbers with more digits. Models were trained on 1-8 digit addition. The baseline models were trained without intermediate scratchpad steps.

Używane były tu modele wstępnie wytrenowane na stronach WWW (korpusach „ogólnych”), które następnie dotrenowywano na danych z brudnopsisem lub bez.

Trening „from-scratch” z odpowiednią tokenizacją dałoby się zrobić na mniejszych modelach).

Transformery z brudnopisem. Ewaluacja wielomianów

```
Input:  
Evaluate -7*x**2 + 7*x + 5 at x = 1  
  
Target:  
<scratch>  
-7*x**2: -7  
7*x: 7  
5: 5  
</scratch>  
total: 5
```

Figure 4: Example of polynomial evaluation with a scratchpad. Each term in the polynomial is computed separately and then added.

Table 1: Results for polynomial evaluation task. Scratchpad outperforms direct prediction whether using fine-tuning or few-shot.

| | Few-shot | Fine-tuning |
|-------------------|--------------|--------------|
| Direct prediction | 8.8% | 31.8% |
| Scratchpad | 20.1% | 50.7% |

Transformery z brudnopskiem. Wykonywanie programów

DIRECT EXECUTION PREDICTION

Consider the following Python function:

```
def remove_Occ(s,ch):
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    for i in range(len(s) - 1,-1,-1):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    return s
```

→ Large Language Model → assert remove_Occ("PHP", "P") == "H"

Fill in the ??? below:

```
assert remove_Occ("PHP", "P") == ???
```

SCRATCHPAD TRACING

Consider the following Python function:

```
def remove_Occ(s,ch):
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    for i in range(len(s) - 1,-1,-1):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    return s
```

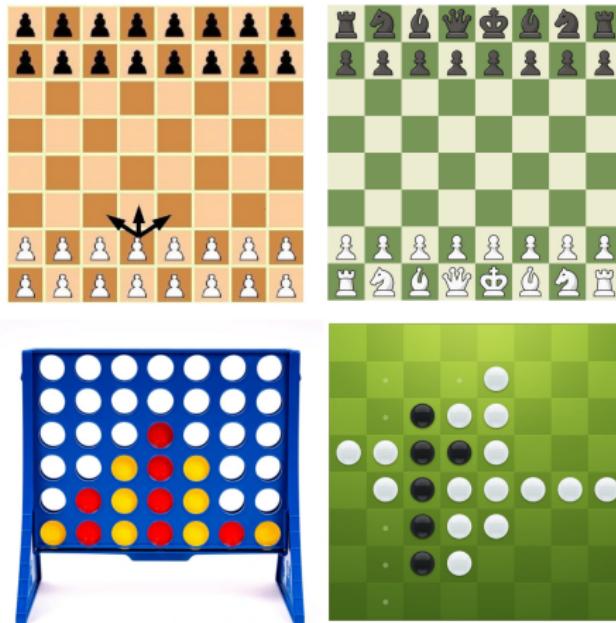
```
output = remove_Occ("PHP", "P")
```

What is the execution trace?

→ Large Language Model →

```
state: {}
line: def remove_Occ(s,ch):
state: {"remove_Occ": "<callable_object remove_Occ>"}
line: state: {"ch": "P", "s": "PHP"}
line: for i in range(len(s)):
state: {"ch": "P", "s": "PHP", "i": 0}
line: if (s[i] == ch):
state: {"ch": "P", "s": "PHP", "i": 0}
line: s = s[0 : i] + s[i + 1:]
state: {"ch": "P", "s": "HP", "i": 0}
line: break
state: {"ch": "P", "s": "HP", "i": 0}
line: for i in range(len(s) - 1,-1,-1):
state: {"ch": "P", "s": "HP", "i": 1}
line: if (s[i] == ch):
state: {"ch": "P", "s": "HP", "i": 1}
line: s = s[0 : i] + s[i + 1:]
state: {"ch": "P", "s": "H", "i": 1}
line: break
state: {"ch": "P", "s": "H", "i": 1}
line: return s
state: {"remove_Occ": "<callable_object remove_Occ>"}
"output": "H"
```

Gry planszowe



Jeżeli dla ciągów ruchów umiemy przewidywać kolejny ruch i to tym samym umiemy grać w daną grę (okazuje się, że całkiem dobrze).

Czy autoregresywny model językowy może grać sensownie w szachy?

Może?

Czy autoregresywny model językowy może grać sensownie w szachy?

Może?

Czy autoregresywny model językowy może grać sensownie w szachy?

Może?

- Może się nauczyć otwarć (lepiej niż ludzie)

Czy autoregresywny model językowy może grać sensownie w szachy?

Może?

- Może się nauczyć otwarć (lepiej niż ludzie)
- Nawet bardzo proste modele mogą grać lepiej niż losowo (również poza otwarciem)

Czy autoregresywny model językowy może grać sensownie w szachy?

Może?

- Może się nauczyć otwarć (lepiej niż ludzie)
- Nawet bardzo proste modele mogą grać lepiej niż losowo (również poza otwarciem)
- (semi)-unigramowy model $P(\text{move}|n)$, gdzie n is numerem ruchu może grać lepiej niż losowy (bo?)

Published as a conference paper at ICLR 2023

EMERGENT WORLD REPRESENTATIONS: EXPLORING A SEQUENCE MODEL TRAINED ON A SYNTHETIC TASK

Kenneth Li*

Harvard University

Aspen K. Hopkins

Massachusetts Institute of Technology

David Bau

Northeastern University

Fernanda Viégas

Harvard University

Hanspeter Pfister

Harvard University

Martin Wattenberg

Harvard University

ABSTRACT

Language models show a surprising range of capabilities, but the source of their apparent competence is unclear. Do these networks just memorize a collection of surface statistics, or do they rely on internal representations of the process that generates the sequences they see? We investigate this question in a synthetic setting by applying a variant of the GPT model to the task of predicting legal moves in a simple board game, Othello. Although the network has no a priori knowledge of the game or its rules, we uncover evidence of an emergent nonlinear

OthelloGPT

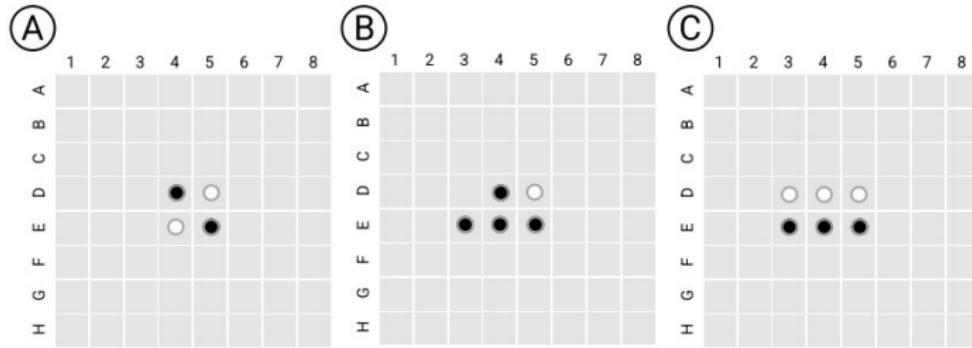


Figure 1: A visual explanation of Othello rules, from left to right: (A) The board is always initialized with four discs (two black, two white) placed in the center of the board. (B) Black always moves first. Every move must flip one or more opponent discs by outflanking—or sandwiching—the opponent disc(s). (C) The opponent repeats this process. A game ends when there are no more legal moves.

Popatrzmy na demo z przedmiotu SI (i zauważmy, jak dynamiczna jest plansza)

OthelloGPT – wnioski

- Przewiduje poprawne ruchy z b. dużą dokładnością – nauczył się zasad

OthelloGPT – wnioski

- Przewiduje poprawne ruchy z b. dużą dokładnością – nauczył się zasad
- Tworzy wewnętrzny model planszy (ew. na tablicy o **probingu**)

OthelloGPT – wnioski

- Przewiduje poprawne ruchy z b. dużą dokładnością – nauczył się zasad
- Tworzy wewnętrzny model planszy (ew. na tablicy o **probingu**)

Czy umie grać sensownie? Autorzy nie stawiali tego pytania...